Toolpath verification using set-theoretic solid modelling

Wallis, Andrew Francis

Award date:
1991

Awarding institution:
University of Bath

Toolpath Verification using Set-Theoretic Solid Modelling

Submitted by Andrew Francis WALLIS

for the degree of PhD

of the University of Bath

1991

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Acknowledgements

Firstly, I would like to thank Dr. John Woodwark who was the original supervisor of the research project described in this thesis. Dr. Woodwark introduced me to the topic of solid modelling and provided much encouragement and inspiration both for the work described here, and also in the wider field of solid modelling. I would also like to thank my colleagues in the Manufacturing Group of Bath University School of Mechanical Engineering for their patience, and to Dr. Adrian Bowyer in particular for his advice during the final stages of the writing of this thesis. The work was originally funded by the SERC and Delta CAE Ltd. (now DeltaCAM), and I gratefully acknowledge this.

SUMMARY

The manufacture of components using Computer Numerically Controlled machine tools is widespread in engineering industry. The checking of toolpath descriptions for these machines before component manufacture begins is important if costly mistakes are to be avoided. This thesis describes a technique for toolpath verification using Solid Modelling. The basis of the work is the generation of a spatially divided model of the component that results from subtracting a model of the volume swept by the cutter from a model of the component blank. An algorithm is presented for the control of the spatial division process. Techniques of picture generation and model interrogation using raycasting are described.

Also presented is an algorithm for the input of facetted set theoretic models from two dimensional contours.

Contents

CHAPTER 3: Generating and Storing (Set-Theoretic) Solid models in a divided state

CHAPTER 4: Input to a Solid Modelling System

CHAPTER 5: Toolpath Verification

CHAPTER 1

Introduction

Computer Representation of Shape

One of the main applications of computing in engineering is the representation of the shape of engineering components. There are many ways of representing shape [1], although most techniques may be classified into one of three categories. One of the earliest uses of computer techniques to represent the shapes of engineering components were Computer Aided Draughting systems. The simplest draughting systems store a list of lines and vertices in two-dimensions that are equivalent to a manually produced engineering drawing. Although the computer-based system offers many advantages over the manual approach in terms of ease of drawing creation and modification, it does not (necessarily) contain any additional information.

Three-dimensional draughting systems were developed as extensions of these two-dimensional systems. Vertices and lines in three dimensions are stored and two-dimensional drawings are generated by projecting the three-dimensional data into a two-dimensional plane. As in the case of the 2D systems, the only information stored are the lines and vertices. The term *wire-frame* is often applied to such systems since the data-structure contains the same information that could be used to generate a physical model of the component from pieces of wire. It is important to note that there may be no representation of the form of the faces of the component, and certainly no information as to which parts of the model are solid, and

which are not. This means that inconsistent and ambiguous models may be constructed, one well-known example (shown in figure 1.1) consists of two nested cuboids that can be interpreted as a block with a chamfered hole oriented in one of three directions.

The second category of modelling systems are the *surface modelling* systems, which, as is implied by the name, are used to model the surfaces of components. They were developed mainly for describing surfaces that were be cut by numerically-controlled machine tools. The surfaces modelled by such systems are often curved (sculptured surfaces) and the approach used to model them is to approximate the surface by a large number of patches [2]. Whilst these systems model the surface form of a component, they still do not store any information as to solidity. Also, there are often few or no checks to ensure that the surfaces used to described a component are consistent. Hence it is possible to describe shapes that cannot exist in the real world. Nevertheless, surface modelling systems are often capable of modelling a wide range of surface geometries, and with suitable care, have a wide range of engineering and computer-graphics uses.

Another three-dimensional representation is the polygonal face-model. These represent the surface of a model by a list of faces, each of which is a (possibly convex) polygon. This may be regarded as an incomplete Boundary-representation (B-rep). As such it is difficult to ensure consistency in such a model, and the generation and maintenance of such structures may prove difficult. However they are of importance partly because many algorithms for hidden-line and hidden-surface elimination have been based on such models.

The third type of modelling scheme models not only the surfaces of a component, but also contains information as to its solidity. These systems are referred to as *solid modelling* systems. The solidity information enables solid modelling systems to ensure that their internal models are represent real shapes. It also allows them to answer queries such as the calculation of the volume or surface area of a model. Whilst these systems have clear advantages over the surface modelling systems, there are also a number of penalties associated with them when compared to wire-frame and surface modelling schemes. Firstly, the internal data-structures may be more complicated due partly to the additional data that has to be stored, and also to the requirement to ensure that the model is unambiguous and consistent. Secondly the algorithms needed to allow users to generate solid models are generally more complicated. Similar considerations apply to the generation of output from the modelling system. Lastly, the requirement to ensure consistency in the data-structure means that many solid modelling systems are less generous in the range of surface types that they can model than surface modelling systems.

The completeness of the information stored in a solid modelling scheme means that such models may be used as the basis for a wide range of applications (which are listed later). The absence of any ambiguity in the models allows many of these applications to be performed automatically.

Applications of Solid Models

The principal engineering applications have been in the areas of component design, analysis and manufacture. The ability to represent three-dimensional shapes in

complete and consistent manner, plus the capability of generating consistent sets of two-dimensional drawings makes solid modelling systems very useful for design purposes. Finite element meshes for thermal or mechanical stress calculations may by generated automatically from solid models [3]. The automatic generation of toolpaths for numerically controlled machine tools from solid models [4], [5], [6], [7] is a current research topic.

Representation Schemes for Solid Modelling

There are a number of ways of implementing a solid modelling system; a good explanation of seven different techniques is given by Requicha [8]. Two techniques have emerged as being suited to most modelling requirements. These are *graph-based* (boundary-representation or B-Rep) modellers; and *set-theoretic* modellers, sometimes referred to as Constructive Solid Geometry (C.S.G.) systems.

The B-rep systems use a data-structure that stores the faces, edges and vertices of a model. Links between data items store the relationship between individual faces edges and vertices. One technique of arranging these links is to store bi-directional pointers between each edge in the model, the vertices at the ends of the edge, the faces that lie on either side of the edge, and those other edges that share a vertex and a face with the edge. This 'winged-edge' data structure which is shown in figure 1.2 was first used by Baumgart in the GEOMOD [9] modelling system.

Set-theoretic modelling systems store a model as a hierarchical set-theoretic combination of simpler objects [10]. At the lowest level, they are based on a set

of 'simple' shapes that are the *primitives* of the system. The internal data-structure of these models is far-less complicated than that for the B-rep modellers, consisting of a simple set-theoretic description of the model in terms of its primitives. Set-theoretic models are dealt with in greater detail in Chapter 2.

Either representation scheme allows other, non geometric, information to be stored in the model. Typically attributes may be attached to model primitives or surfaces representing colour or texture. For engineering applications tolerances may be useful, although maintaining consistent tolerances information over an entire model [11] may be difficult. Other attributes may be used to record the material that a component is made from, or to provide links to a company data-base of components.

B-rep and Set-Theoretic Modelling Schemes Compared

B-rep models require a complicated data-structure to store all of the point, edge and face information together with the various pointers that link them together. Care has to be taken when making incremental changes to B-rep models so that invalid models are not generated. This is especially the case if the user is allowed to locally 'tweak' the data-structure since he could, for example, move a vertex through a face of the model. Another related problem is that the geometric data has to be stored to a high level of precision, and care has to be taken to avoid inconsistencies resulting from arithmetic rounding errors. Some errors may be detected by applying Euler's rule, although dummy edges may be required with curved faces to maintain the data-structure. Since edge information is stored in the

model, B-rep modellers have to be capable of calculating and representing the edges that result from the intersection between any pair of primitive shapes handled by the system.

Membership testing (described more fully in the context of set-theoretic models in Chapter 2) is not particularly efficient with B-rep modellers since it requires intersection tests between a vector and each face in the model. The fact that edges are present in the structure does mean that B-rep modellers are more easily linked to draughting systems than set-theoretic systems; also simple line-drawings are easily generated from the model. The B-rep data-structure contains topological locality, although this does not guarantee spatial locality, indeed spatial coherence within the B-rep model may be difficult to check.

Set-theoretic modellers have several advantages when compared with their B-rep counterparts. Some of these, such as conciseness and rigor have been extensively documented by the University of Rochester Production Automation Project [12] in the context of their PADL system. The data-structure for a set-theoretic model is very simple, especially when compared to the B-rep structure, it is also directly related to the textual input languages often used to describe models. Due to the nature of the representation, set-theoretic models are always valid. Membership testing for set-theoretic models is simpler than for B-rep models, requiring a point comparison with each half-space, rather than an vector-face intersection test. The accuracy problems sometimes associated with B-rep modellers tend not to arise with set-theoretic modellers since the models are stored in an 'unevaluated' state (although consideration may be required as to the closed or

open nature of the primitives and set-theoretic operators used [12]). Constructing B-rep models from set-theoretic models is a relatively simple process, indeed the input system for B-rep modellers may use set-theoretic combinations; conversely there are at present no algorithms for the reverse process of converting from B-rep to set-theoretic representations for general three-dimensional models.

For these reasons a set-theoretic modelling scheme is used for the work reported in this thesis. Chapter 2 describes various techniques used for the evaluation of set-theoretic models. One problem associated with this evaluation is that it requires a large amount of computation; a technique which has shown much promise in reducing this computational load is that of spatial division and model pruning. The generation and storage of spatially divided models is discussed in Chapter 3.

Chapter 4 describes an algorithm for the generation of set-theoretic models from 2-dimensional outlines consisting of straight lines.

Toolpath Verification

The increasing use of computer numerically controlled (CNC or NC) [13] machine-tools by engineering industry, and the increase in the complexity of the toolpaths handled by such machines has lead to a requirement for checking of the toolpaths prior to machining. Methods of performing this verification are described in Chapter 5. The use of computers for toolpath verification has become very widespread although many commercial verification packages simply display the path followed by the tool centre, rather than the shape that would result from

machining. More sophisticated computer-based techniques are reviewed. One of the most sophisticated approaches is to use solid modelling techniques to represent the shape of the material being machined.

A Toolpath Verification System based on Solid Modelling

Chapters 6, 7 and 8 described a toolpath verification system that uses spatially divided set-theoretic modelling techniques to allow the checking of toolpaths for n.c. machine tools. The generation of the set-theoretic model from the toolpath is described in Chapter 6. Chapter 7 describes how this model is spatially divided. The interactive technique used to interrogate this model, allowing the user of the system to check the toolpath is presented in Chapter 8.

For practical reasons it was decided to restrict the range of toolpaths that would be processed. Toolpath verification for turning applications is a relatively simple task, as the tool movements are essentially in only two dimensions. It is hence suitable for a range of simple plotting techniques and has already been tackled by a number of systems. It was decided to investigate the more complicated problem of toolpath verification for 3-axis vertical milling machines. The technique is clearly also applicable to the simpler problem of verification for lathes. It is also capable of being extended for the verification any toolpath where the volume swept by the tools is capable of being represented by implicit polynomial surfaces of reasonable degree.

The Modelling Schemes used for this Project

During the period of research two versions of the toolpath verification system were written. They both use the same overall technique, but differed in the choice of model primitive, and in the details of the division and raycasting algorithms. In the first system, models are constructed using planar half-spaces. All non-planar surfaces are approximated by facetting using a number of planar half-spaces. Rather than use a fixed number of half-spaces, which would be wasteful for cylinders with small radii and result in large facets for cylinders with large radii, a limit is placed on the maximum facet size. The number of facets used to model a given cylinder (or part of a cylinder) is determined from the radius of the cylinder, such that the maximum deviation of the faceted model surface from the required surface is less than a predefined maximum acceptable error. Hence a compromise can be reached between the number of half-spaces in the model and the maximum surface deviation error. In the following chapters this modeller is referred to as the *faceted system.*

The second verification system uses a geometric modelling scheme based on general polynomial (in x, y and z) half-spaces. A modelling system based on these primitives was written by the author and Dr. J.R. Woodwark as part of a project investigating blends for set-theoretic models. The system is fully described in reference [14] . The choice of primitive means that a wide range of surface geometries may be modeled exactly without the need to approximate them by facetting. In the following chapters this is referred to as the *polynomial system.*

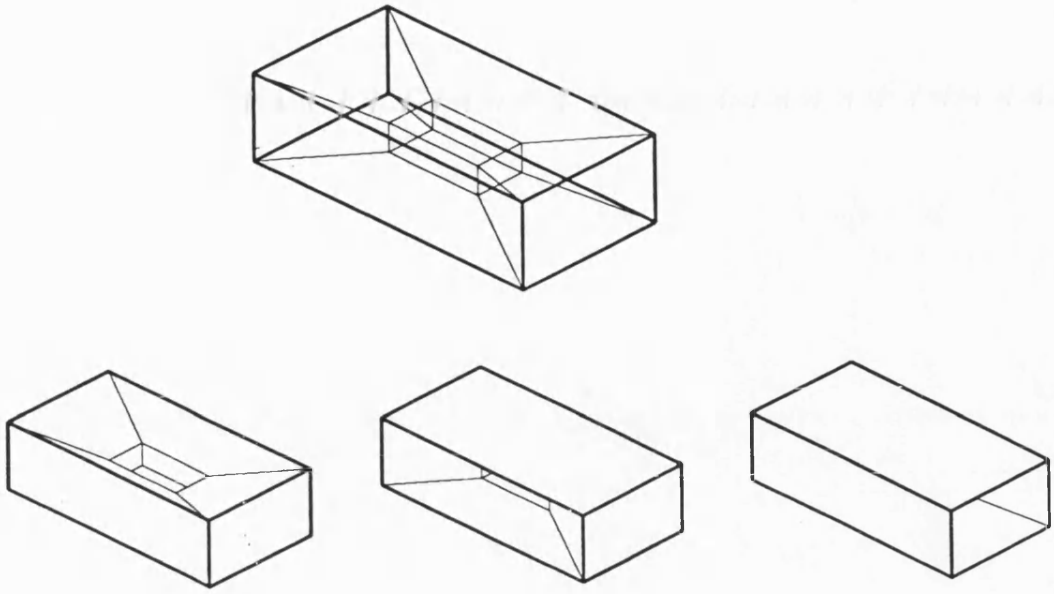

Figure 1.1 Example of wire-frame ambiguity

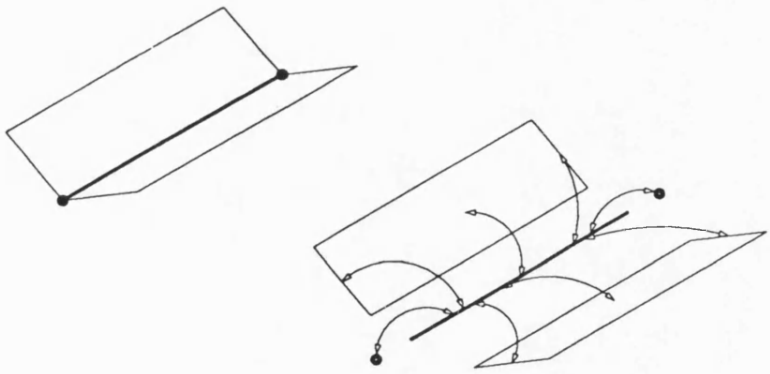

Figure 1.2 Winged-edge data structure

CHAPTER 2

Evaluation of Set-Theoretic Models

Set-Theoretic Models

In a set-theoretic modelling scheme objects are defined as the set theoretic combination of simpler sub-objects, which may themselves be constructed from even simpler objects. The definition thus lends itself naturally to a hierarchical structure, described more fully below. The operators used to combine the sub-objects are the set-theoretic operators *union*, *intersection* and *difference* (shown in figure 2.1). Obviously some objects are needed to form a starting point for this combinatorial process. These objects are called the primitives of the modelling system.

The range of different primitives available to the user varies between systems as does the technique used for their internal representation. One approach is to provide a collection of bounded objects: for example, unit cubes, cylinders and cones. These primitives may be stored as individual boundary-file models. Other systems use unbounded solids (directed-surfaces or half-spaces). The simplest half-space is the planar half-space which divides space into two regions, one defined as solid the other as air, on either side of a plane. More complicated half-space geometries can also be used, and the set of primitive half-spaces can be extended to include, for example, cylindrical spherical and toroidal half-spaces.

It should be noted that the difference operator is not strictly required. The set-theoretic algebraic identity,

$$\mathbf{A} \perp \mathbf{B} \equiv \mathbf{A} \cap \overline{\mathbf{B}}$$

means that the difference operator may be replaced by an intersection operator and the complement of the object to be differenced. If **B** above is not a primitive then de-Morgans Rules may be used to generate the complement in terms of the constituents of **B**:

$$\overline{(\mathbf{B1} \cap \mathbf{B2})} \equiv \overline{\mathbf{B1}} \cup \overline{\mathbf{B2}}$$

$$\overline{(\mathbf{B1} \cup \mathbf{B2})} \equiv \overline{\mathbf{B1}} \cap \overline{\mathbf{B2}}$$

Note that in order not to restrict the range of shapes that may be modelled, the complement of each primitive must also then be a primitive of the system. This is not a problem for systems whose primitives are half-spaces, but may present difficulties for those based on bounded primitives.

A Data-structure for Storing Set-Theoretic Models

The hierarchical model definition is most conveniently stored as a tree. The non-leaf nodes in the tree contain the set-theoretic operators. Leaf nodes contain the primitive elements (which will be referred to in future simply as elements) of the system. In order to construct a model it is necessary to be able to place the primitives into the required positions and orientations, and at the correct sizes. This may be achieved by applying geometric transformations to a set of unit-sized primitives defined at the origin before combining them with the set-theoretic operators.

If the primitives are bounded solids then it is usual to include geometrical transformations in the tree. Geometric operators such as rotations, transformations and scaling are included as non-leaf nodes in the tree. In such a model the leaf nodes contain either references to primitives, or arguments to these geometric operators. Figure 2.2a shows a simple set-theoretic model built from a unit sized cuboid primitive.

When the model is constructed from half-spaces it is convenient to transform the half-spaces before including them in the tree. This avoids the need to include the geometric operators in the tree. It is also computationally more efficient since transformations are performed at model generation rather than at model evaluation time. The additional storage requirement for the transformed half-spaces is more than compensated for by the simplification in the tree structure. This pre-transformation is not so desirable for models based on bounded solid primitives since, if done, it precludes them from being stored in a canonical form. Figure 2.2b shows the same model as shown in figure 2.2a constructed from planar half-spaces.

If the primitives of the system are half-spaces then they can simply be defined using implicit polynomial functions (see figure 2.3). For example, a planar half-space may be defined as the set of points (x,y,z) such that

$$ax + by + cz + d \leq 0$$

where a, b, c, and d are constants; and curved half-spaces may be defined by performing arithmetic operations on these planar half-spaces, for example a spherical half-space or may be defined in terms of three planar half-spaces that intersect at

its centre:

$$sphere := hs1^2 + hs2^2 + hs3^2 - radius^2$$

When the model is evaluated (for example to generate a picture of it) it is necessary to be able to evaluate the primitive elements. The form of the primitive evaluation will depend on the required model evaluation. In the case of creating pictures of the model the intersection of lines and primitives, or the classification of points with respect to primitives may be required.

If the evaluation process is to first generate a B-rep model from the set-theoretic description, and then perform the evaluation on the B-rep model it is advantageous to use primitives represented by B-rep models since all of the faces and some of the edges and vertices of the model are contained in the primitives and hence do not have to be generated at evaluation-time.

Output from Set-Theoretic Models

One of the most common forms of output from a solid modelling system is graphical. Solid modelling systems are used mainly in disciplines such as engineering and architectural design, where the traditional method of communicating information is via technical drawings. In most applications the objects being modelled are at least partially manufactured by people. In all such cases some form of graphical output is required. Even if this is not the case then some form of graphical output is desirable in order to verify that the model is correct. If the model is created by a person, rather than as the output of a computer program, then some form of graphical feedback is essential.

Graphical output from solid modelling systems can be categorised into two classes: line drawings and continuous-tone pictures, examples of which are shown in figures 2.4 and 2.5.

Line Drawings

Line drawings display the edges of the model. In the case of set-theoretic models, the edge information is not held explicitly and so has to be calculated before the edges can be drawn. Edges may be formed along the intersection line of two or more half-spaces, or, in the case of a model with bounded primitives, along the intersection line of the faces of two or more primitives and also along the edges of the primitives themselves. If a simple set-theoretic modelling scheme is used in which the model is stored in a single tree, then, in order to generate those edges resulting from primitive-primitive intersections, each primitive in the model must be compared with every other primitive to see if they intersect. When such an intersection is found, the intersection curve can be added to a list of candidate edges. If the two primitives that form the edge have the same local surface normal, then the surface is locally flat and a candidate edge should not be created. (Such flatness may be detected by using a more complicated version of the membership testing technique that is described later in this chapter.) Only some of these edges will be real edges. Those edges that do not lie on the surface of the object must be removed from the list. This may be achieved using a membership test (explained in the next section).

The result of the membership test applied to a point is only certain to be valid for those points that have the same classification relative to all of the primitives in the model. Hence, before a candidate edge can be tested, it must first be divided into segments bounded by the model primitives (see figure 2.6). Each segment can then be classified by applying a membership test to its mid-point. Those edges whose mid-points are in solid or air are eliminated from the list of candidate edges.

Note that if the system is based on bounded primitives, such as polyhedra, or if the primitives have curved faces, such as tori, then the sets of edges created by primitive-primitive intersections may be complicated.

A complication arises when the model contains non-planar surfaces. To be realistic the line drawing has to include *horizon lines*. These occur where the surface normal changes from pointing towards, to pointing away from, the viewer. These lines are not related solely to the geometry of the model, but also to the projection and viewing parameters used. It is also sometimes useful to plot additional lines that lie on the surface in order to display the curvature of the surface. For example, in the case of a cylinder lines may be plotted parallel to its axis, or in the case of a sphere, meridian lines may be added.

Tilove [15] describes an algorithm for the generation of wireframes from set-theoretic models based on B-rep primitives. Once the list of real edges (ie those that lie on the surface on the model) has been formed they can then be displayed using the required projection. The projections most often used are orthogonal (parallel), isometric and perspective.

It is often desirable not to display the edges that would not be seen by the viewer. Their removal may be performed in two stages. A partial solution is to remove those lines that face away from the viewer. This is a relatively simple process as it only requires checking the orientation of the primitives or half-spaces that form the edge. If both faces that form an edge are oriented away from the viewer then the edge cannot be seen. In practice, this check may be made at the whilst candidate edges are being generated.

Full hidden line removal requires that each remaining edge be compared with the model to check if the complete edge, or part of it, is hidden from view. Many algorithms have been developed to perform this hidden line elimination task. Five such algorithms (together with five hidden surface removal algorithms) are compared in [16]. Others are described in [17]and [18]. All of the algorithms are, as presented, suitable for hidden line elimination for polygonal face models although they could be used with B-rep solid models.

One characteristic of all of these algorithms, as well as Tilove's wire-frame generation algorithm is that they avoid the need to compare each edge with every face or primitive in the model. To do this they exploit the *coherence* present either in the model, or in the picture being generated from it.

Membership Tests

Membership tests are used to find out whether a given point lies inside or outside the model, or on its surface. To so classify a point it is first compared against each primitive or half-space in the model; the point will either be inside or outside the

primitive or half-space, or will lie on its surface. The contributions of each primitive (solid, air or surface) are combined using the operator tree. Starting at the leaf nodes in the tree and working back to the root, each non-leaf node in the tree may be classified by combining the classifications of each of its son nodes using a set of rules. This results in a single classification for the root node which is also the classification of the point. A simple set of rules is given in table 2.1, and an example of a membership test in figure 2.7. Note that the surface-normal directions of primitives classified as *surface* may be important if points are not to be in classified incorrectly.

Continuous tone Pictures

Continuous tone pictures display the surfaces of the model rather than its edges. The surfaces are shaded or coloured according to the colour of the primitive that contains the surface, modified by a suitable lighting algorithm. The elimination of hidden surfaces is clearly essential (these pictures are meaningless without it). Although all of the surfaces that are present in the model are contained in the set-theoretic description, they are stored in an unevaluated state. Thus before the surfaces can be plotted the model has to be processed in order to find which parts of the surfaces of the primitives form real surfaces of the model. (Only those model primitives that appear in the picture need to be tested, a fact that the VOLE modeller [19]uses to advantage.)

One approach to generate continuous tone pictures is to create a boundary model (or simply a polygonal face model) from the set-theoretic representation.

The faces in the model can then be coloured, projected and plotted. Sutherland *et al*'s paper describes five algorithms for plotting polygonal face models with hidden surfaces eliminated.

One of these algorithms, Warnock's [20], projects the polygons onto a rectangular window and then recursively divides it into a number of smaller rectangular regions. At each stage of division the polygons are classified against the regions as to whether they lie completely outside, cover the region, or partially cover it. Division terminates when either 1) no polygons lie within the region, or 2) a single polygon covers the region lying in front of all other polygons, or 3) the region is smaller than some limiting size (the size of a single pixel in the case of a picture being created on a raster-scan display). In the second or third case, the region is coloured in according to the front-most polygon

Other algorithms [21] and Bronsvoort [22] work in scan-line order and are based on sorting the polygons into ascending *x*, *y*, and *depth* order and calculating when, for each scan-line, each polygon starts and ends, since it is at these points that the foremost polygon changes.

Raycasting

An alternative technique is that of 'ray-casting'. This technique may be used to create perspective (and other) views on raster-scan displays. For each pixel in the picture a ray vector is generated. The position and direction of each vector depend on both the position of the pixel on the display screen, and the projection in use. (A good explanation of the raycasting process, and applications of

raycasting is given by Roth [23].)

The simplest (and most natural) projection is the perspective view. A view point and a vector representing the direction of view are first defined in the coordinate system of the model. A notional regular grid of points having horizontal and vertical resolution the same as that of the image to be generated is placed between the view point and the model in a position and orientation such that the viewing vector intersects its centre. Each point in the grid represents a pixel in the image. The ray vector for each pixel is the vector that passes through the view point and the point in the grid for that pixel (see figure 2.8). Views in parallel projection may be generated using a similar technique, except that all the ray vectors are parallel to the viewing direction and pass through points on the grid.

A simple and naïve method of processing each ray vector for a set-theoretic model *directly*, rather than working from a set of polygonal faces is given in figure 2.10. The intersections of the ray with all primitives in the model are found, sorted, and then tested in order until either a real surface intersection is found, or until all intersections have been tested. If no intersections are found or none of the points lie on the surface then the ray does not intersect the model and the pixel may be coloured in a background shade.

It should be noted that this simple scheme is not suitable for anything but the simplest of models. Its performance is severely effected by model complexity (described later in this chapter), and, in practice, more complicated processing methods need to be used that have better order, and are hence more suited to complicated models containing large number of primitives, especially if the primitives

contain high degree surfaces. Techniques for increasing the efficiency of raycasting are described later in this chapter.

Roth uses a scheme that generates an ordered list along the ray vector of regions that lie inside and outside each primitive in the model. These regions are combined using the set-theoretic model definition to generate a list of regions for the entire model. The start of the first region that lies inside the model corresponds to the first surface hit by the ray. The efficiency of his raycaster is increased by the use of 'enclosures'; rectangular regions that surround each primitive (see the section on boxing tests). For each ray, only those primitives whose enclosures are intersected by the ray need to be considered. Enclosures for non-leaf nodes in the set-theoretic model tree may be found by combining enclosures for the siblings of the node.

Bronsvoort *et al* [24] describe two methods of improving the performance of raycasting set-theoretic models. Firstly by using scan-line intervals instead of enclosures. These intervals are generated by projecting each primitive onto the screen and calculating the limits of each interval on the current scan-line found. The 'ray intersects enclosure' test is replaced with the simpler 'point within interval' test. The reported reduction in CPU time for scan-line intervals over boxing enclosures is 5 to 15 percent. The technique is not suitable for casting secondary rays, as required for generating shadows for example. The second method is to cast rays on a coarse grid (every 4 pixels for example) and then recursively refine the grid when this appears to be necessary. This can clearly reduce the number of rays cast, but may result in errors in the picture.

Shading the Picture

When generating a continuous-tone picture, the pixel corresponding to any surface may be coloured according to the colour of the surface modified using a lighting model. One simple model uses Lambert's Cosine Law [25] for diffuse illumination: the colour for the pixel is generated by multiplying the red, green and blue intensities for the surface colour by a factor proportional to the cosine of the angle between the surface normal at the intersection point and a vector to a light source (figure 2.9). More complicated lighting models also allow for specular reflections and for the generation of shadows.

If a picture is to be created by ray-casting, then these effects are relatively easy to create. Pixels that are in shadow may be detected by, having found a surface point, casting a second ray from the point to the light source. If the ray intersects the model then the surface point will be in shadow. Mirrored surfaces may be modelled in a similar manner. Whilst the pictures generated for the toolpath verification system described later do not usually contain these effects, the technique of generating secondary rays is used.

The Problem of Model Complexity

Although the techniques used for generating line drawing and shaded images differ, they both potentially suffer from three problems related to the 'complexity' of the model. These problems, which are always encountered when evaluating set-theoretic models, are:

- The problem of primitive complexity,
- The primitive combinatorial problem,
- The model 'size' problem.

The first two problems are discussed here, the third after the section on facetting.

Primitive Complexity

Some evaluation processes, such as the calculation of the intersection between a vector and a primitive when raycasting, are affected by the geometric complexity of any individual primitive in the model. If the surfaces of the primitives or half-spaces are planar, or are second, third or forth degree polynomials, then the intersection of the ray vector with the primitive may be calculated directly. Higher degree surfaces require the use of iterative techniques. These are often time consuming, and for any fixed precision of arithmetic, are limited to a maximum degree of surface that may be handled without errors occurring.

Primitive Combination

Other processes, such as the calculation of the edges of a set-theoretic model or the generation of a boundary file model, are effected by the interaction between pairs of model primitives. Edge generation requires that the curves of intersection of the primitives in the model be found. If there are no restrictions on the positioning on the elements of the model, then intersections may occur between any two primitives, in any relative positions. Since intersections may occur between any two

types of primitive, the number of types of intersection curves, and hence the number of different solutions to be coded, is equal to half the square of the number of different primitives. Thus for systems with a large number of primitive types, the number of different solutions to be coded becomes very large. The calculation of these intersection curves for any but the simplest of primitive shapes is non-trivial. It is costly both in the amount of coding required to implement it, and also in computation time when the edges are being generated.

Some early systems, PADL-1 for example [26] reduced the effect this problems to some extent by restricting the user as to the orientation of primitives in the model. (If there are restrictions in the positioning of primitives then they should, as a general rule, be handled by the input processor, and not left to the user, otherwise the results are not predictable.)

It may be noted that ray-casting avoids problems caused by the interaction between primitives (such as occur when the generating the edges of a model) and so is less effected by the geometric complexity of the model primitives than other image generation techniques.

Facetting

One way of avoiding the first two problems is to replace all curved surfaces in the model with a number of planar surfaces. This technique is called facetting. In a bounded-primitive model this may be achieved by replacing primitives incorporating curved surfaces with polyhedral primitives. In a half-space model, curved half-spaces may be replaced by a number of planar half-spaces. Examples of

facetted models are shown in figure 2.11.

Facetting may be performed at the input stage by either the user or the modelling system when defining the model, or by the modelling system at the evaluation stage. A modelling scheme based on planar half-spaces is attractive for several reasons.

If half-space facetting is performed at the input stage then the data structure of the model is simplified as it contains only one type of primitive, the planar half-space. The evaluation of such a model is also simplified. For example, the calculation of edges requires only the calculation of the intersection of two planes.

Another advantage of facetting is that all surface geometries may be modelled, albeit approximately. This limited accuracy of the facetted model is obviously undesirable in some applications, such as the automatic generation of toolpaths from a model. For other applications especially those whose output is approximate in nature, such as the calculation of model volume using 'Monte-Carlo' techniques, or applications whose output is of a graphical form it is not necessarily a problem. Obviously the larger the number of facets used, the smaller the discrepancy between the model and the object being modelled. However, if the number of facets becomes very large then the advantages of the simplified data-structure and coding may be outweighed by the sheer amount of data. If the level of facetting is controlled by the user then it is possible for him to balance the needs of accuracy, storage and computation overheads.

In general the number of facets does not become excessive if the model contains only planar and singly curved surfaces. This is the case for a large number of engineering components, including all two-and-a-half dimensional objects.

The use of facetting does, however, increase the number of elements in the model. This highlights the third model complexity problem, which results from the relationship between the number of primitive elements in a model and the computational load incurred during its evaluation. The problem is now considered.

Model Size

The valid, although naïve, algorithms for the evaluation of set-theoretic models described earlier in this chapter require comparisons between all of the primitives in model. This is because of the unevaluated nature of the representation scheme, and will therefore apply to all evaluation processes (examples include the generation of wire-frame, polygonal face models or B-rep solid models, the generation of line or continuous-tone pictures, or the calculation of the volume of the model).

The naïve edge generation algorithm first compares each element in the model with each other element in order to find the candidate edges. The membership tests for each candidate edge requires it to be compared with each element again. Thus the computational load for the generation of edges is proportional to the cube of the number of elements in the model. The computational load to create a boundary model from the set-theoretic definition involves many of the same steps and will also be proportional to the number of elements cubed.

The generation of a raycast picture from a set-theoretic model in also affected by the number of elements in the model. The intersections of the ray vector with each element in the model must be calculated and sorted. Then, on average, half of these must be membership tested before a real surface point is found. Thus the computational load varies with the square of the number of elements in the model.

If the model contains a large number of elements, the time taken by any evaluation process will grow to be very large. It is obviously desirable to reduce this effect of model complexity. To achieve this reduction in evaluation time for large models, it is necessary either to reduce the number of comparisons to be made, or to reduce the time taken to perform them.

Increasing the Efficiency of Model Evaluation

Boxing Tests

One method of increasing the efficiency of model evaluation is to classify each element in the model spatially prior to its evaluation. In a model based on bounded primitives this may be done using boxing tests.

For each element in the model, a surrounding enclosure is constructed. When comparing two elements in the model to see if they intersect, the surrounding polyhedra for each are first compared. If they are found not to overlap, then the more detailed tests required to calculate the intersection curves need not be applied. This reduces the computational load required to evaluate the model. Similarly, when ray-casting, each ray can be tested to see if it intersects the enclosure prior to calculating its roots with the primitive.

The enclosures that are simplest to use are cuboids aligned with the coordinate system (a two-dimensional example is shown in figure 2.12). These may be calculated by finding the minimum and maximum of each primitive along each coordinate axis. The cuboids may easily be compared using a minimax test of their coordinates. Testing for intersections of a ray vector with the cuboid and generating cuboids for non-leaf nodes by combining other cuboids is also easily done.

This simple boxing technique has two drawbacks. Firstly, if the elements in the model are not positioned orthogonal to the axes, then the surrounding boxes will include large amounts of space not occupied by the primitive. Secondly, the effectiveness of the test is greatest when the model elements are mainly disconnected. If the model consists of inter-penetrating elements then it is of limited value.

The first of these drawbacks may be overcome, to some extent at least, by using more sophisticated forms of surrounding enclosure (general convex polyhedra for example). These could be constructed to fit closer to the outlines of the primitives. However this would impose a greater computational load both to calculate the polyhedra and also to process them. Roth [23] suggests using spherical enclosures, but concludes that in most cases they perform less well than cuboids.

It should be noted that boxing tests are not suitable for models using unbounded primitives. There is however an alternative spatial classification technique that is applicable to such models: spatial division.

Spatial Division

The technique of spatial division and model *pruning* has been used in a number of modelling systems developed at the University of Bath. The sole primitive of the earlier of these systems (VOLE [27], [19], SODA [28], [29], and DORA [30]) is the planar half-space. Non-planar surfaces are approximated by facetting. The models are defined by a list of planar half-spaces and a tree structure comprising references to these half-space and set-theoretic operators.

One of these systems, VOLE-2 [19] was developed for the purpose of investigating the problem of the computational time required to generate continuous tone shaded pictures from set-theoretic models, and in particular the relationship between this time and model complexity (as measured by the number of half-spaces in the model).

The technique used by VOLE-2 to process a model is one of divide and conquer. A cuboid object-space is defined surrounding the model and oriented with the viewing direction. This object-space is recursively split into a number of sub-spaces. A sub-model is generated for each sub-space by pruning the original set-theoretic definition of the model to the sub-space. This creates a simpler sub-model applicable to that sub-space alone. When a given level of sub-model complexity is reached the division is terminated and the sub-model is evaluated. The details of this evaluation, which generates a part of the picture, which is valid for the sub-model, are given in reference [19]. One important result obtained from the VOLE-2s+2 modeller was that the relationship between model complexity and evaluation times is better than linear.

A more recent modelling system (DODO) written by the author uses general polynomial half-spaces rather than planar half-spaces. This modeller also uses spatial division and pruning to increase the efficiency of image generation. It is this modeller that forms the basis of the toolpath verification system which uses polynomial half-spaces.

Model Pruning

The pruning process is an important part of the spatial division technique. In order to create the pruned sub-model for a sub-space each half-space in the model is compared with the sub-space. If the half-space passes through the sub-space then the half-space is included in the sub-model for the new sub-space. If it does not then the half-space is replaced with either *air* or *solid* in the sub-model depending on which the half-space contributes to the sub-space. The following rules may be applied in order to simplify the sub-model. **X** represents any single half-space or part-model.

$$Air \cap \mathbf{X} \rightarrow Air$$

$$Air \cup \mathbf{X} \rightarrow \mathbf{X}$$

$$Solid \cap \mathbf{X} \rightarrow \mathbf{X}$$

$$Solid \cup \mathbf{X} \rightarrow Solid$$

Having classified each half-space against the sub-space, the rules may be applied to the set-theoretic tree that describes the sub-model, working from the leaf-nodes back to the root. At each stage, if either son of a node is *air* or *solid*, then the corresponding rule may be applied to prune out one of the sons. In gen-

eral, as the size of the sub-spaces is reduced, fewer half-spaces pass through it, and so the complexity of the pruned sub-model is reduced.

Table 2.1

Rules for simple Membership Test

Operator	Classification of first primitive	Classification of second primitive		
Union		*Air*	*Solid*	*Surface*
	Air	Air	Solid	Surface
	Solid	Solid	Solid	Solid
	Surface	Surface	Solid	Surface
Intersection		*Air*	*Solid*	*Surface*
	Air	Air	Air	Air
	Solid	Air	Solid	Surface
	Surface	Air	Surface	Surface
Difference		*Air*	*Solid*	*Surface*
	Air	Air	Air	Air
	Solid	Solid	Air	Surface
	Surface	Surface	Air	Surface

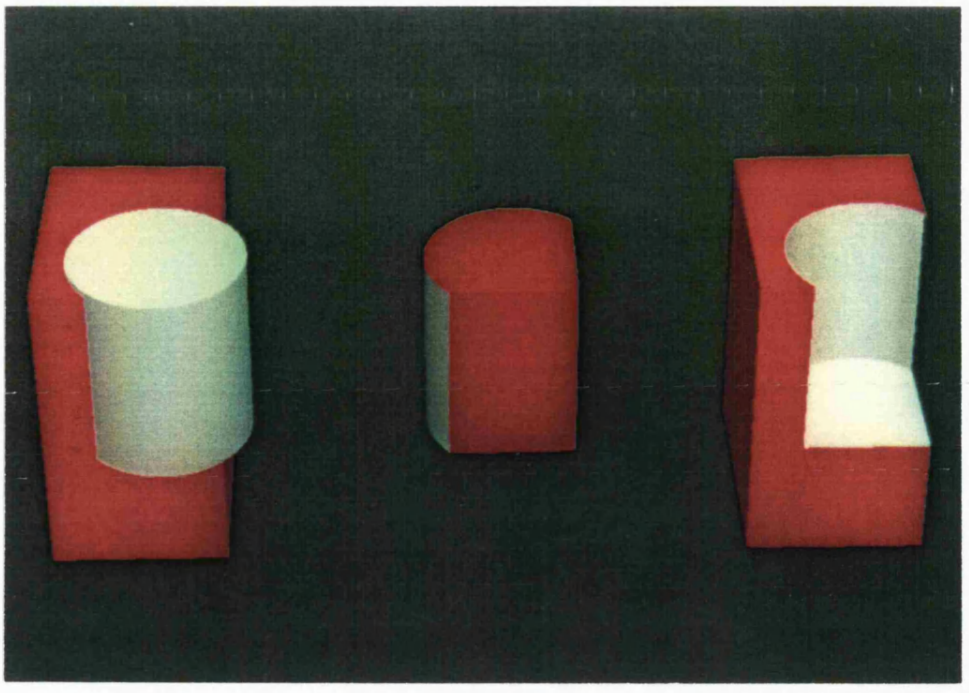

Figure 2.1 The set-theoretic operators

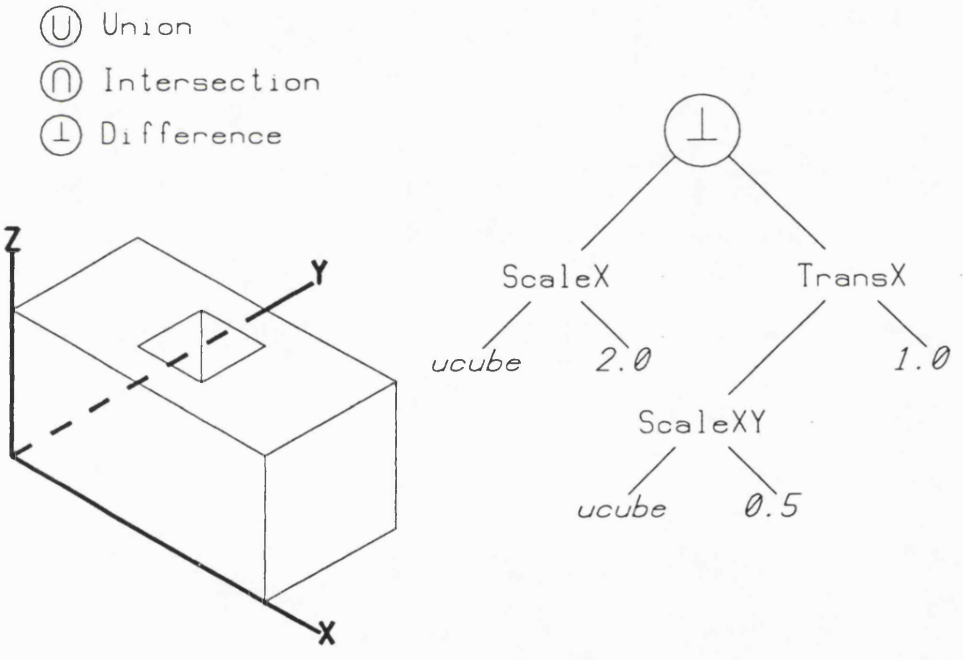

Figure 2.2a A primitive based set-theoretic model tree

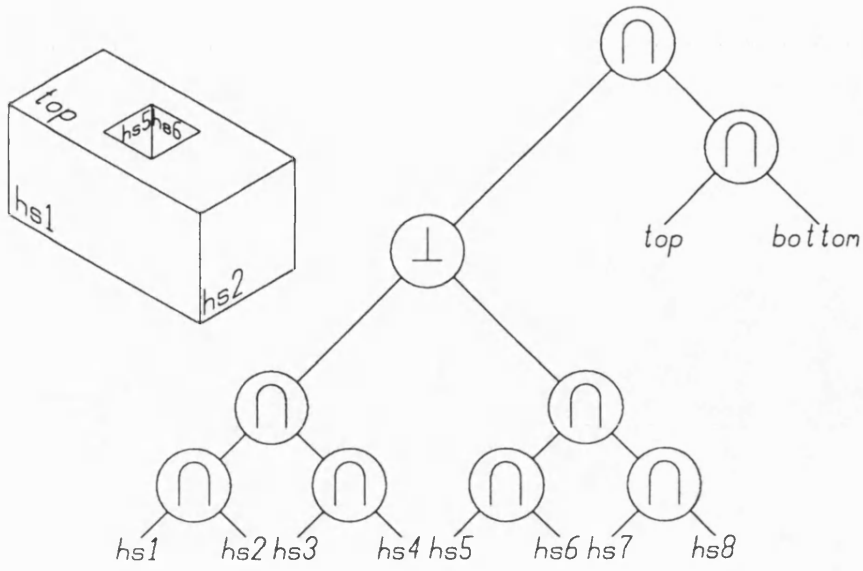

Figure 2.2b A half-space based set-theoretic model tree

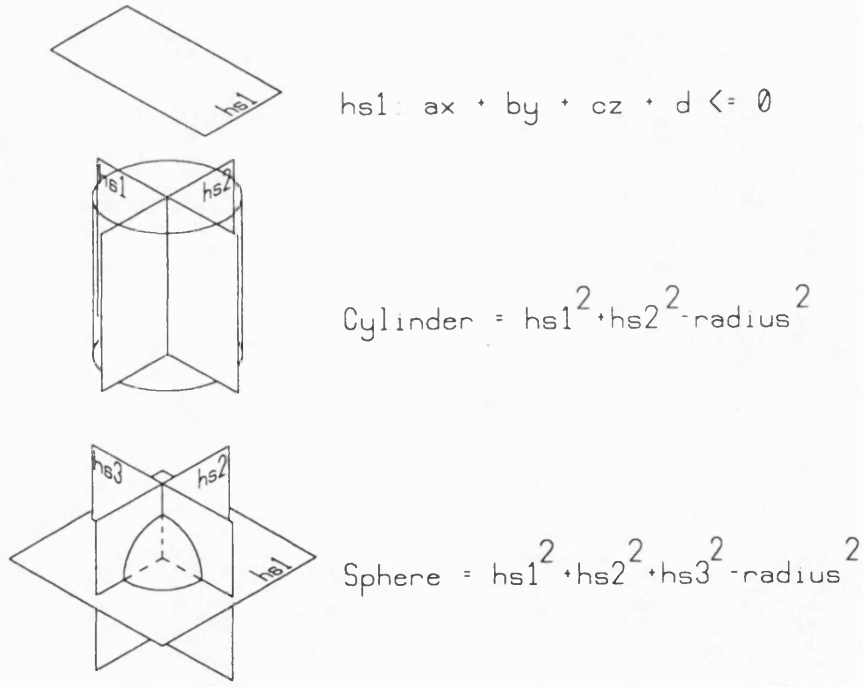

Figure 2.3 A 'polynomial' cylinder and sphere

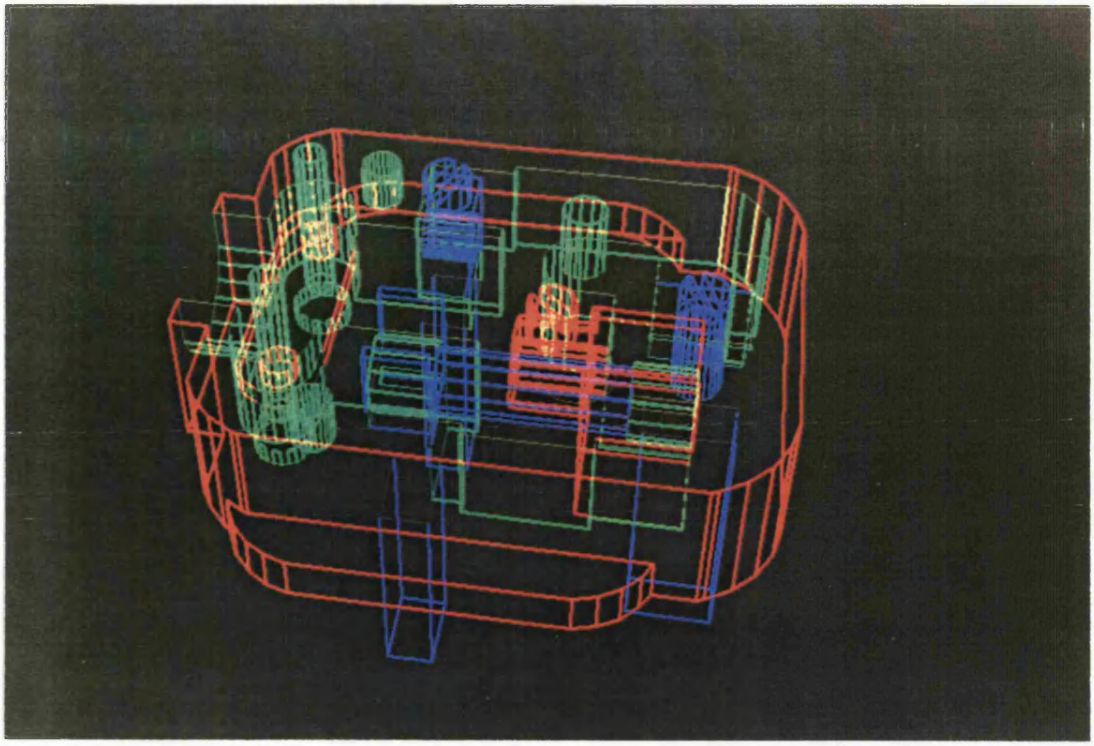

Figure 2.4 An example of line-drawing output

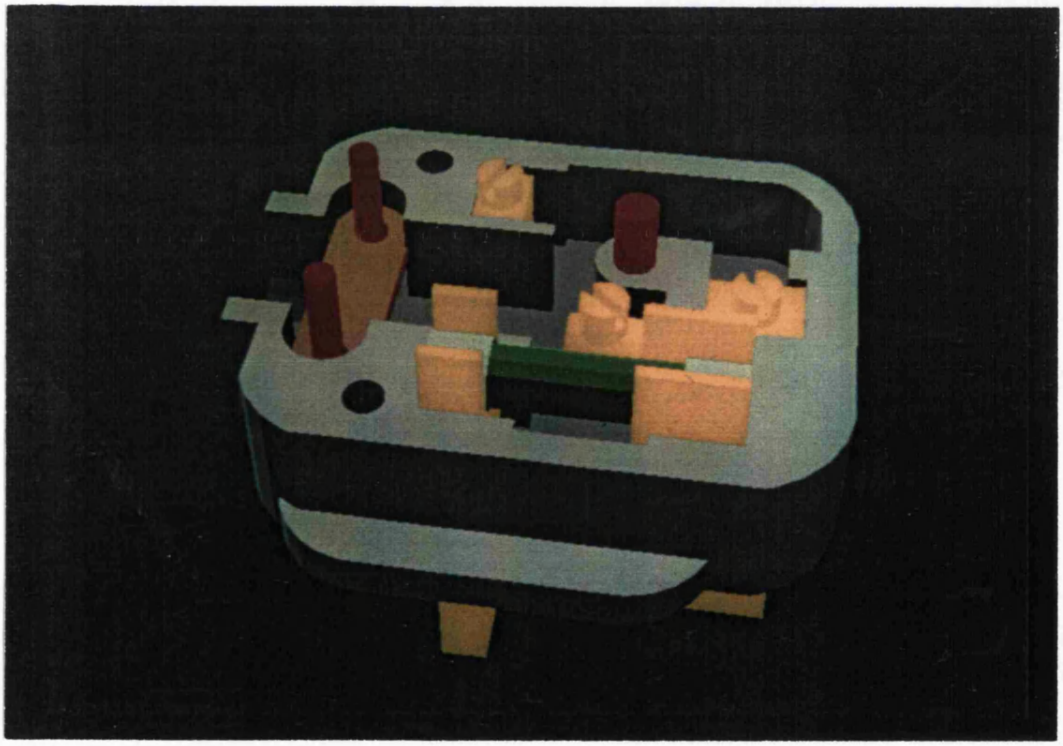

Figure 2.5 An example of continuous-tone output

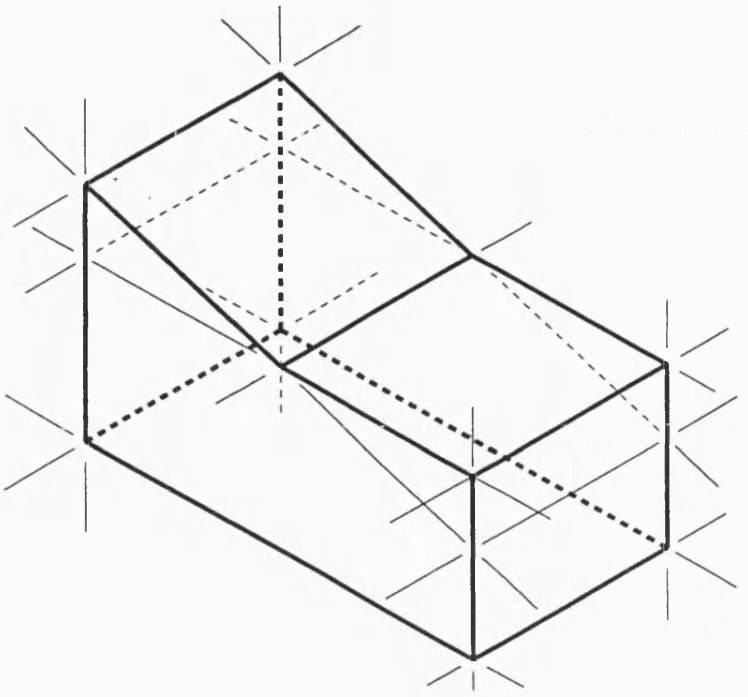

Figure 2.6 Generating Line drawings, segmenting edges

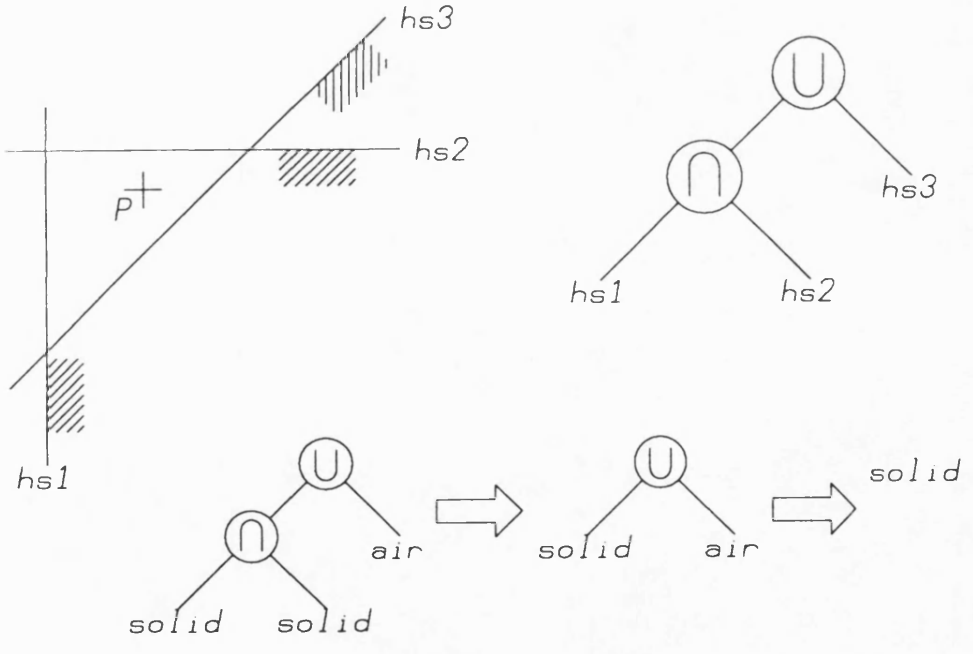

Figure 2.7 Membership testing on a point

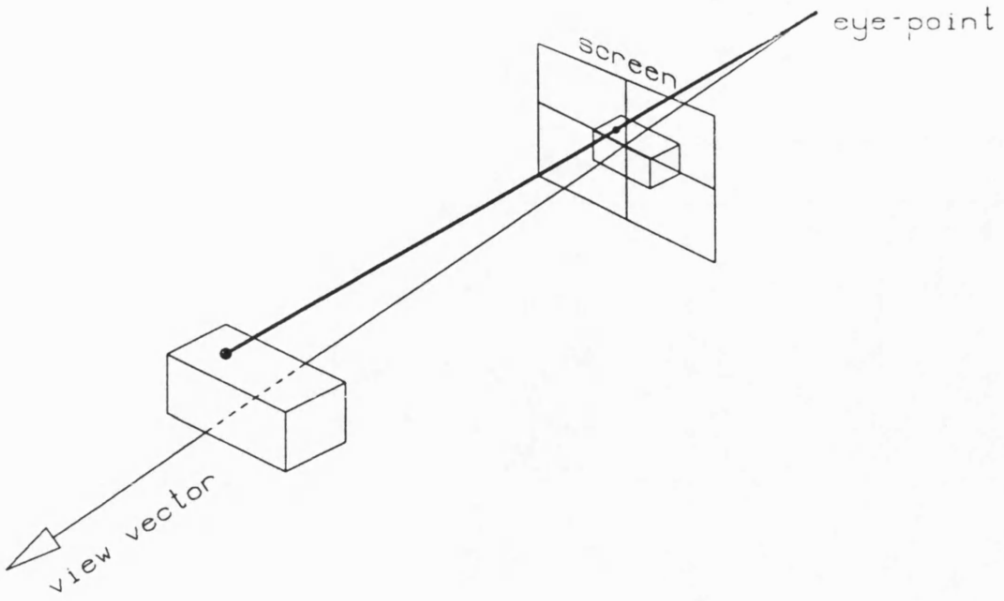

Figure 2.8 Raycasting

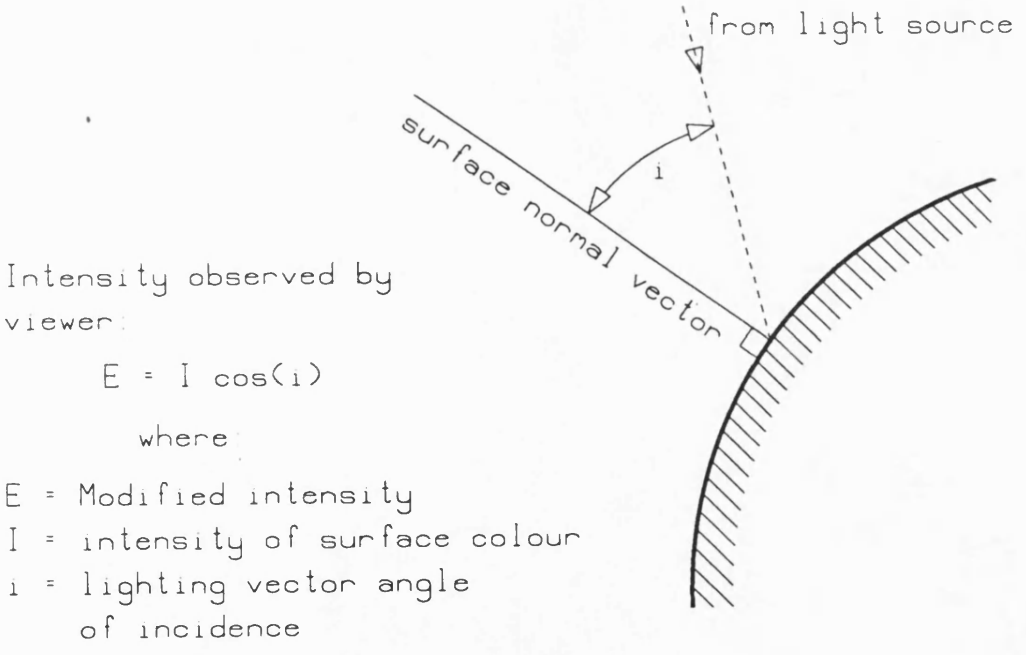

Figure 2.9 Determining surface colours: Lamberts cosine law

```
/* A procedure for casting a single ray into a set-theoretic model. */

procedure raycast()
        {
        Zero list of intersection points
        For each primitive
                {
                Generate intersection points of ray with primitive.
                Add points to list of intersection points.
                }
        Sort point list
        Repeat
                {
                Get next point in list
                Classify point using membership test
                }
        Until(surface-point found or list empty)
        If(list empty)
                Colour pixel as backgound
        Else
                Colour pixel according to primitive colour and lighting model
        }
```

Figure 2.10 A Raycasting procedure

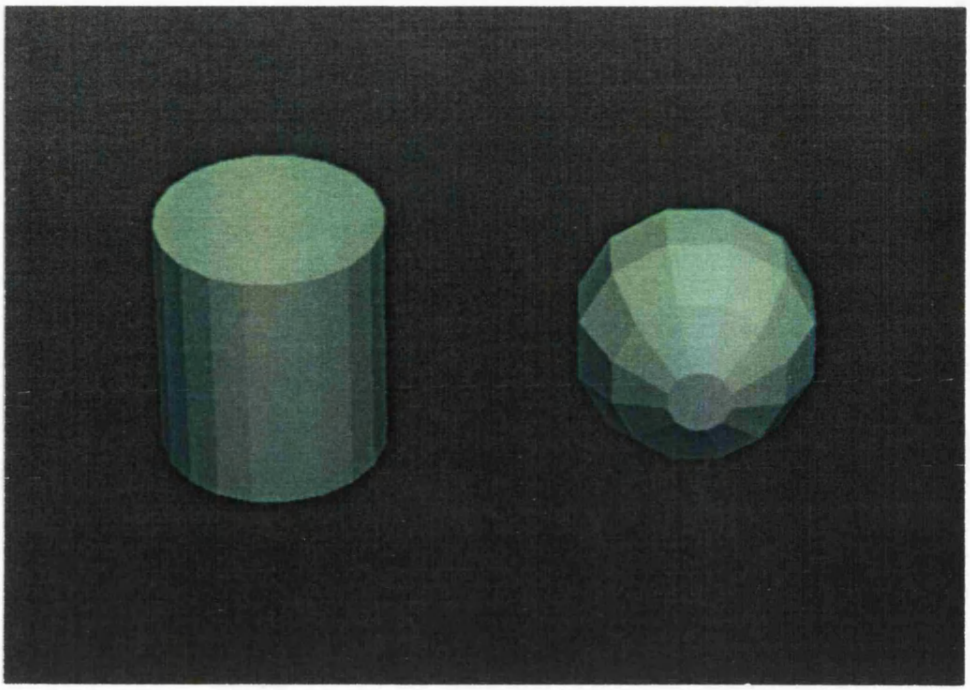

Figure 2.11 A Facetted cylinder and sphere

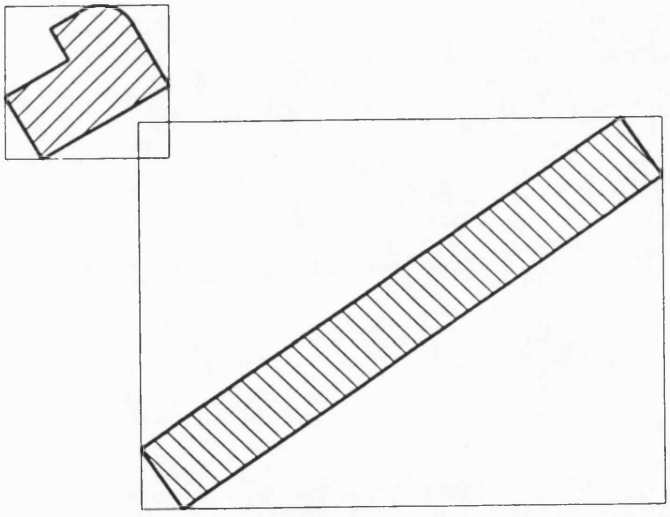

Figure 2.12 Boxing tests

CHAPTER 3

Generating and Storing (Set-Theoretic) Solid models in a Divided State

In the last chapter some of the problems encountered in the evaluation of set-theoretic models were raised, together with some techniques for minimising their effect. One technique, as used in the VOLE [19] and DORA [30] geometric modelling systems, is the use of spatial division and model pruning. In VOLE the division process was used as a method of generating a single picture from a set-theoretic model, and was oriented to achieve this efficiently. The divided structure was used and then discarded. This chapter will consider a more general form of spatially divided model. Rather than use the divided model once only, the concept of the divided model as a spatially divided data structure is introduced. Different strategies for dividing models are discussed, together with techniques for generating a spatially divided model stored in a tree structure.

Object Division and Spatial Model Division

The main objective of generating a divided model is to reduce the computation time required to 'evaluate' the model. This evaluation may be, for example, the generation of one or more views of the model or the calculation of its mass properties. In all cases the computation load incurred when evaluating the model is related to the complexity of the model (as described in the previous chapter). If

division of the model into a number of sub-models is to result in a reduction of this computation load, then the complexity associated with model evaluation must also be reduced. This may be achieved if the complexity of each sub-model is reduced. When evaluating a model that has been divided into a number of sub-models there is an additional load associated with accessing the sub-models which may also need to be considered.

It is important to differentiate between the two types of division: spatial division and object division.

Object division involves splitting the model itself into a number of sub-models. This may be achieved by referring to the primitives used in the model together with its set-theoretic definition. Tilove [15] and [31] describes how this 'structural' or 'representational' locality can be used to reduce the amount of computation required when generating a new B-rep model from a modified set-theoretic model. By identifying the local region within complete set-theoretic model definition where changes have taken place, the overall computational load (as compared to that required if using the complete description) is much reduced, since only part of the complete tree needs to be considered.

In a spatially divided model, the volume containing the model is divided into a number of sub-spaces. The half-spaces, primitives or sub-models that comprise the various parts of the model are classified as to whether they occur within each of these sub-spaces. The sub-model may then be pruned to form a simpler model valid for the sub-space. Figure 3.1 shows an two-dimensional example of a spatially divided model which, although it divides the *object-space* into only three

suitable from models based on half-spaces. When a *query* is asked of a spatially divided model, the spatially divided structure may be used to quickly localise the sub-space(s), and hence the primitives upon which more detailed examination is required (a point membership test for example).

Now consider the advantages and disadvantages of each scheme. Model division is the better strategy for answering queries that are model based, such as requesting the spatial location of a part of the model. It is not so good, however, at answering queries that are spatially based. In order to find which parts of the model occupy a given space, for example, the list of sub-models will have to be searched (unless an auxiliary spatial data-structure is maintained containing such information). Also, if the model input definition is used as a basis of the division, the actual division will depend on the way in which the model was constructed rather than on its actual shape.

A problem with using structural locality is identifying the local region(s) of the model that need to be considered for a particular operation. This is particularly so since there need be little connection between the spatial positioning of primitives in a model and their position in the set-theoretic model definition.

The spatially divided model is more suitable for applications were the queries are spatially oriented. Most solid modelling applications, examples of which include the generation of pictures of a model, the calculation of the mass properties of a model and the automatic generation of toolpaths from a model, have such requirements.

For the toolpath verification system the queries (which will be considered in detail later) are of a spatial nature; also, the structure of the model will usually be unknown to the user of the system since it will have been constructed automatically; the model itself will be based on half-space primitives and will be potentially very large. For these reasons a spatial division strategy is used.

The Requirements of a Spatially Divided Data-structure

Now consider the requirements of a spatially divided set-theoretic model for use as a divided data structure. The most important consideration is that downstream processes that use the model should be able to use the divided structure in order to evaluate the model efficiently. The size and format of the structure should be such that the generation time is not excessive. Also the structure should not be so large that the memory requirements to store it become excessive. Each element in the spatially divided structure has to contain the size and position of a region in space (stored either explicitly or implied by its position in the data-structure), and a model that is valid for that space.

Regular (grid) Division

The simplest form of spatial division is to divide the object-space evenly into a number of regions or sub-spaces, each of fixed size positioned in a regular grid pattern. Owing to the even nature of the division the location of the sub-space within the data-structure that contains any given position in space is easily calculated. This is advantageous at the evaluation stage, since, for a given point in

space, the sub-space that contains that point may be accessed directly. Also, the neighbours of any sub-space may be found directly. The TIPS modelling system [32] uses an even grid against which primitives are classified, and, when performing any calculation at a location within a grid cell, those primitives that lie completely outside that cell may be effectively ignored. Any form of even division suffers from the disadvantage that (unless the complexity of the model is also evenly distributed throughout the model) the divided structure is inefficient in that some parts of the model are overdivided whilst other regions would benefit from further division. Also, the size of grid structure quickly becomes large since it must be divided down to the smallest required resolution.

For some processes the large number of cells in a regular grid can have an adverse effect on the efficiency of the process. An example occurs when raycasting using such a data-structure which requires a partial traversal of the structure. In this case many adjacent cells may have to be traversed whose contents are identical and which have been created because of complexity elsewhere in the model.

Tree Structures for Divided Models

These disadvantages of regular division can be overcome if the sub-spaces are of variable size. Then the size of the sub-spaces can be adjusted to suit the complexity of the model in that region. Allowing variable size sub-spaces does however increase the problem of finding which sub-space contains a given point in space.

One solution is to use a tree structure. The sub-spaces that contain the model (and correspond to those in the even division scheme described above) form the

leaf nodes in the tree. The root node represents a region that surrounds the object to be modeled (the *object space*). Other nodes in the tree are organised such that the sub-space represented by each non-leaf node is identical to the union of the sub-spaces represented by its son nodes. Thus at any depth in the tree the total volume of the object space is represented. The spatially divided model of figure 3.1 can be simply stored as the tree-structure shown in figure 3.2.

The tree structure can be used to access the model. As stated above the combined volume for the sons of each non-leaf node represent the same volume as the node itself. Hence in order to find which leaf node contains the model that is valid for a given position in space, the tree structure can be descended by starting at the root node and then repeatedly testing to see which son of the current node contains the point. The process terminates when a leaf node is reached, and the sub-model contained in the leaf node is valid for the point. This tree descent involves only a few comparisons and hence is very quick to compute. As will be seen later, a similar technique may be used to generate a list of leaf nodes whose sub-spaces are intersected by a vector.

The sub-spaces can be of any shape. As with boxing enclosures, there is a trade-off between the greater level of model pruning that can be achieved with complicated shaped sub-spaces, arbitrary convex polyhedra for example, and the greater level of computation required to access each sub-space (for example, finding which leaf-node sub-space contains a given point). It is usual for the sub-spaces to tessellate the object-space such that they do not overlap, and at any level of division they completely fill the object-space. (These properties are, however

not necessarily required; sub-spaces could, for example, overlap; although care may be required when using such a divided structure.) Sub-spaces are commonly cuboids aligned with the coordinate axes.

An alternative to the tree structure is used by Mäntylä *et al* in their EXCELL [33], [34](extended cell) based modelling system. This approach uses a hierarchical regular grid (called a directory) to point to a binary divided model (called the data part). For regions of the object-space that are accessed using the first-level directory, access is very quick; for regions containing more complicated parts of the model, where several hierarchical levels of directory have to be descended, the computation required to access a data-part entry is similar to that which occur if the model was stored as a simple binary tree.

Oct-trees

It is possible to generate trees of any valency greater than one (although for the general purpose division of three-dimensional space binary and octant division seem most natural). One commonly used division method defines the object-space as a cube surrounding the model and aligned with the axes of the coordinate system. At each stage of division a leaf-node sub-space is split to form eight new cubic sub-spaces by splitting the sub-space along three planes parallel to the axes and passing through its center. The tree structure that results from this process is called an oct-tree (see figure 3.3). The oct-tree structure has the advantage of simplicity; also the size of sub-spaces decreases rapidly, thus generating a shallow tree for any required size of leaf-node sub-space. One disadvantage of oct-tree division

arises from the fact that each sub-space is split into eight sons. It is quite likely that several of these sons will contain the same model. Hence the model is over-divided, resulting in many sub-spaces that are smaller (and more numerous) than is required.

Binary trees

The simplest form of tree structure is the binary tree (see figure 3.4). Binary trees can be used to represent trees of any valency and their simplicity makes them an attractive data-structure, widely used in computing. They may also be used to store a spatially divided model.

One apparent disadvantage of the binary tree structure is that, in order to divide a model to any given level (as defined, say, by the maximum volume of any leaf-node sub-space not containing *air* or *solid*,) a greater depth of tree is required when compared to using a tree of greater valency. When compared to an oct-tree, the depth would be three times as great, and six additional non-leaf nodes are needed in the binary-tree for each non-leaf node in the oct-tree. In practice this does not occur since, except in the unlikely event of the complexity of the model being 'evenly' distributed, the binary-tree is not divided to to same extent as the oct-tree.

In addition to their simplicity, binary trees have the advantage when compared to trees of higher valency that at any stage during their generation a sub-space is split to form only two new sub-spaces. These are then tested independently as to whether they should also be divided. Since only two new sub-spaces are created,

the over-division that can occur when using oct-tree division can be avoided. Such over-division adds to the size of the divided model and is clearly undesirable.

When using raycasting to generate pictures of spatially divided models stored in tree structures it is necessary to partially traverse the tree for each ray cast. All leaf nodes whose sub-spaces are intersected by the ray vector, until the first real surface of the model is reached, will need to be tested. In this application the absence of over-division in the model is especially important, since otherwise the length of the traverse is increased as is the number of leaf-node sub-models that have to be tested. An algorithm for the traversal of a tree such that all nodes whose sub-spaces are intersected by a vector are visited *in order* is given in Chapter 8.

Creating a Binary Divided Model

There are two possible ways of generating a divided-model tree; using either a breadth-first or a depth-first approach. The depth-first approach has the advantage of requiring less storage during the division process. Also the data-structure required to support it is less complicated. Breadth-first division, in which the complete tree is divided (if division is warranted) to ever-increasing levels of division can potentially take advantage of 'global' constraints and goals, such as the maximum allowable size of the completely divided model tree. The technique which is used for toolpath verification system is based mainly on a depth-first method, but uses 'local' breadth-first division. A simple depth-first algorithm for generating a tree structure may be implemented as follows.

To create a tree-structured divided-model, the tree is first initialised to contain a single node whose sub-space is the object space for the model. The tree is then recursively generated by selecting a leaf-node in the tree and dividing its sub-space into a number of smaller sub-spaces, each of which forms new leaf nodes that are sons to the selected leaf node. The sub-model for each new leaf node may be generated from the sub-model for their father by pruning its sub-model to the sub-space for each new leaf-node. Note that sub-models for non-leaf nodes are not required and can be deleted as the tree is generated.

The individual steps in the algorithm will be considered in more detail later in this chapter. The strategy used to detect when the division process has finished, to select which leaf node is to be split, and (in the case of a binary tree) where to position and orientate the split plane, will affect both the time taken to divide the model and also the overall shape of the divided structure.

Ideal Characteristics of a Spatially Divided Model

Ideally we would like the spatially divided structure to be in some way optimised; every leaf node sub-space and sub-model should be in a state where further sub-division will not lead to any simplification in the model. Simplification may, in this context, be defined as a reduction in the computational load imposed on a process that accesses the divided model. In practice the shape and size of the model tree may well be influenced by other constraints, such as the amount of memory available in which to store the divided structure.

In terms of the simple division algorithm outlined above, to generate an optimally divided model it is necessary to decide whether or not a sub-model is best left as it is, or whether further spatial division would lead to a reduction in complexity. The computational load attributed to a sub-space and its sub-model will depend on the process that is to use the divided structure. It will however always be related to the complexity of the sub-model. When arriving at the decision, the computational load required to perform the further division could also be considered.

The physical dimensions of the sub-space should also be considered. The smaller the sub-space, the less frequent it is likely to be accessed by any process using the divided model and hence its contribution to the total computational load is reduced. The exact affect that the size of the sub-space will have on the computational load will again depend on the use of the model.

Now consider a single sub-space that is a candidate for further division. If the sub-space is divided, then the contribution to the total complexity of the divided structure will be the sum of the complexities for the new sub-spaces (plus an allowance for extra computation required to access the greater depth of tree.) Thus a reduction in total complexity will occur only if the the new sub-models are simpler than the original. This simplification will result from the original model being pruned to the new sub-spaces. The degree of simplification (if any) is dependent on the positioning of the constituent parts of the sub-model and also on the set-theoretic operators that join them. Two points should be noted. Firstly, it is not easy in advance to determine the outcome of even a single division stage.

Secondly, a number of divisions may be performed that result in an increase in total model complexity, but one more division may result in an overall simplification.

These two factors make it difficult to use the simple division process described above to generate an efficiently divided model. Ideally a test is required that can be applied to a sub-space (and sub-model) to decide if it requires further division.

Sub-space Testing

Tests for deciding whether a sub-space should be split can be classified as either geometric or non-geometric.

Sub-space Testing: Geometric Tests

The first class comprises tests that investigate the geometric structure of the sub-model. Since, in a set-theoretic model, the data is held in an unevaluated manner, these tests usually involve the generation of geometric entities. A vertex test could be used to generate and count the number of vertices in a sub-model. The division process could then be controlled by limiting the maximum number of vertices allowed within any sub-space. This is the scheme used in the SODA [28], [29] solids database system which is based on planar half-spaces.

Geometric tests can be used to produce a divided model where the distribution of geometric structure is well defined. The main disadvantage with geometric tests is that they can add a large overhead to the division time; for the vertex test, the

computational load to generate vertices in a sub-pace containing n half-spaces is, proportional to n^4.

Sub-space Testing: Non-geometric Tests

Non-geometric tests are based on the unevaluated sub-space and sub-model. One simple test is to count the number of half-spaces in a sub-model and and then continue division until this drops below a predefined limit. Such tests are much quicker to perform than the geometric tests. However, the distribution of real complexity in the resulting sub-spaces may be very unbalanced since there is no direct relationship between the number of half-spaces in a sub-space and the geometric structure it contains. Since these tests do not require the sub-model to be evaluated, the computational load required to perform them in independent of the complexity of the sub-model. They are more suited to applications where the divided structure is used only a few times, where the model is very complicated, or where the application which is to use the divided model is not highly dependent on the actual geometric structure of the model.

Hybrid schemes which are based partly on both types of test can also be used; using a simple half-space count for the initial division stages, and then switching to a geometric test when the number of half-spaces drops below a specified level.

Care must be taken to avoid cases where the division process is non-terminating. If for example the strategy is to continue division until there are fewer than a certain number of half-spaces in a sub-volume, then division will continue indefinitely if there are more than that number of half-spaces passing through

a point. This may be avoided by using additional tests to limit the minimum size of a sub-space.

If the divided model is to be used for a single process then the tests used to control division can be devised such that the measure of complexity is based on the evaluation method used by the process. In this manner, the model division will be optimised for the process. The model divider presented in Chapter 8 uses such a test.

Selecting a Node for Further Division

If a simple test is used which is based on the contents of a single leaf node, then the division technique given previously may be implemented as a recursive algorithm, thereby removing any requirement to select a node. One problem with such an approach is that it may be difficult to control the size of the divided model since it is impossible to tell in advance how large the model will be.

An alternative, which may be used for breadth-first division is to select a node that has the highest computation load associated with it.

Deciding how to Split a Node

The simplest strategy for binary division is to always split the sub-space along its longest side. A non-regular division scheme in which the orientation and positioning of the split-plane for a non-leaf sub-space are determined by the layout of the sub-model primitives can result in much quicker reduction in model complexity, and a more efficient divided structure. This was done by Woodwark in a

modelling system [30] based on planar half-spaces. For modelling systems based on polynomial half-spaces the calculation of the best position for the split-plane is difficult. The model divider used in the toolpath verification system uses a technique that restricts the range of choices to that of the orientation of the split-plane which is located such that it always passes through the centre of the sub-space.

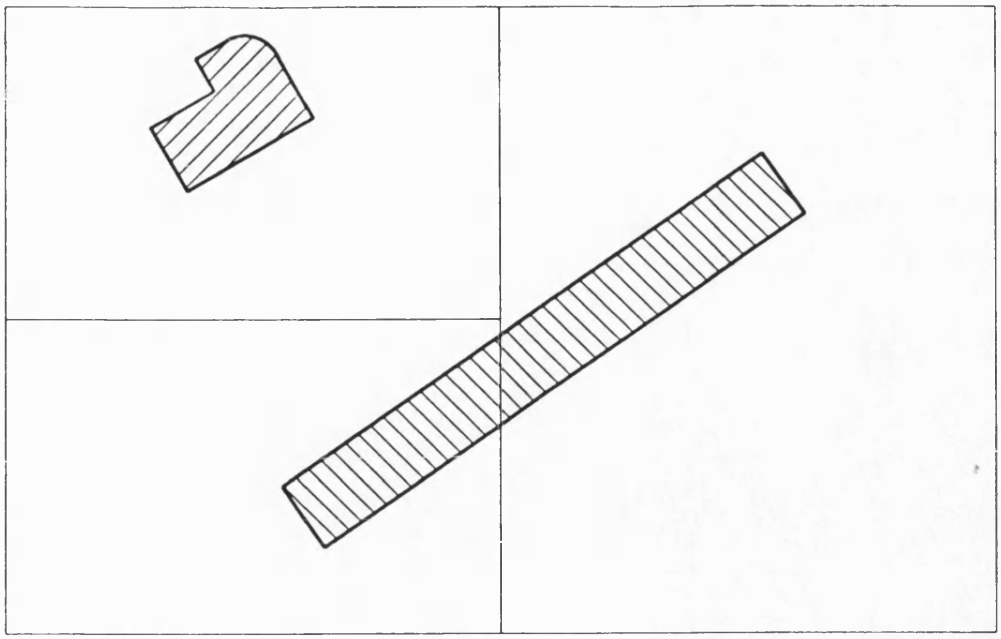

Figure 3.1 A Spatially divided model

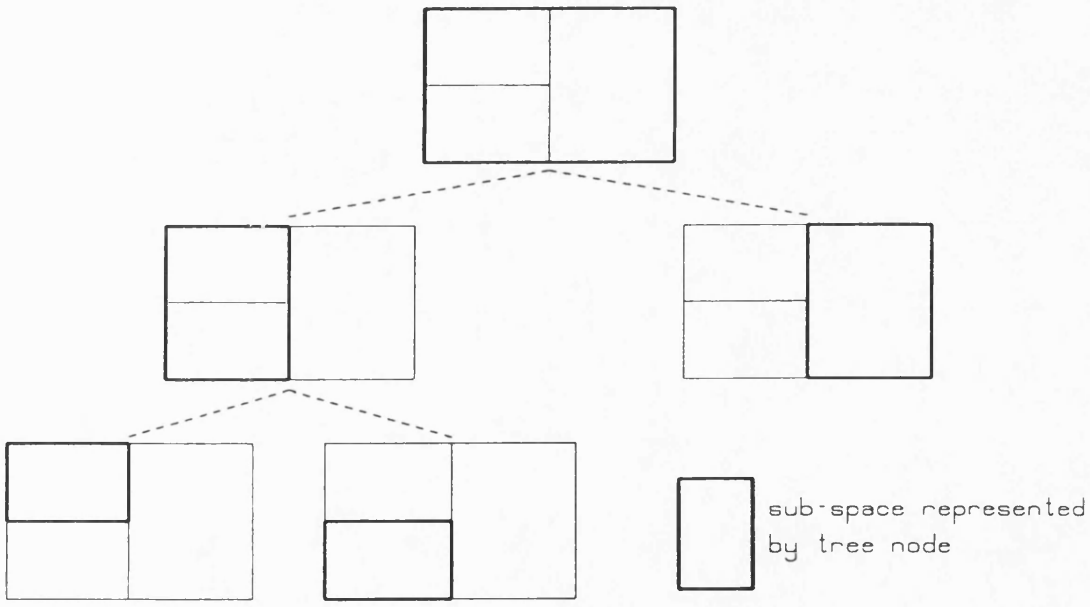

Figure 3.2 Tree-structure for the spatially divided model

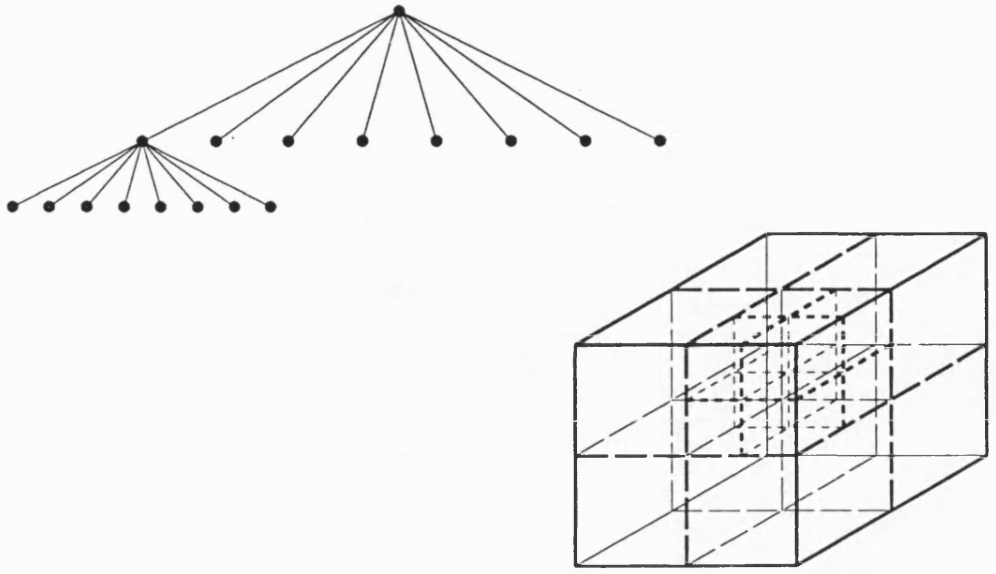

Figure 3.3 Oct-trees

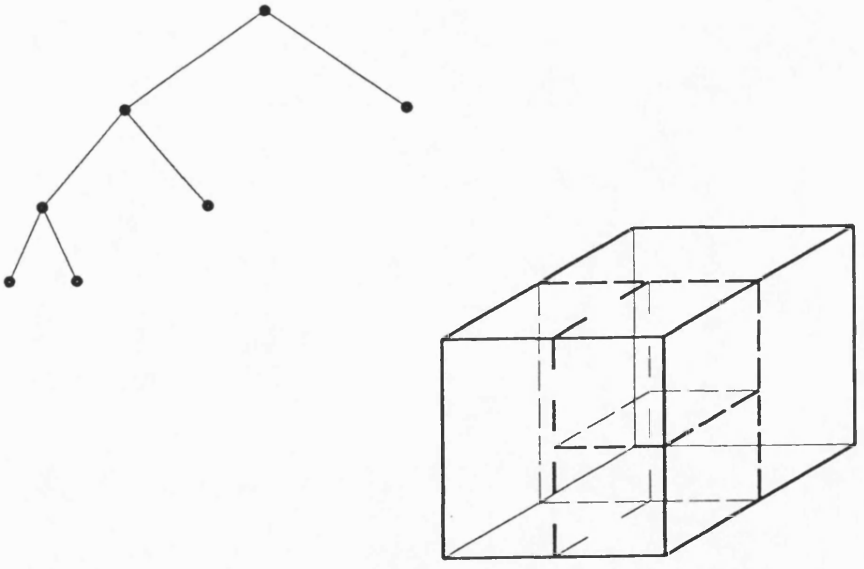

Figure 3.4 Binary trees

CHAPTER 4

Input to a Solid Modelling System

One of the important features of any solid modelling system, certainly from the users' viewpoint, is the techniques provided to define and manipulate shapes. This is especially the case if systems are to achieve widespread use in industry. This chapter will look at the various methods of input used by modelling systems, considering their advantages and disadvantages, and then detail the design and implementation of an input system implemented by the author.

First consider the requirements of the input system. Obviously it must allow the user to describe the model that he wishes to create. Therefore it must allow him to define shapes. The design process often consists of creating objects from other objects. This leads naturally to a hierarchical model definition where models are described as the combination of existing sub-models. Hence the input system must also allow the combination of existing shapes. In order to avoid the repeated definition of similar shapes it must also allow the geometric manipulation of existing models. The input mechanism should be concise and unambiguous.

Alternative Input Techniques

Input methods to solid modelling systems can be divided into three categories:

1) Language input,

2) Graphical input,

3) Computer generated input.

Language Input

Language input was the only method used with several of the first solid modelling systems. In a language input scheme models are described by a written definition. Complicated models may be formed by combining simpler models and there is a strong correlation between the language definition and the design process described above.

Many systems allow the use of parameterised objects. These are sub-models whose dimensions are set by user-supplied parameters. Thus a single parameterised model can be used to create a whole family of similar models. Typically such models are used for standard components such as nuts and bolts. Individual users or companies who make frequent use of specialised families of components may also define them parametrically. In a language input scheme, parameterised objects may be described using functions, with the parameters passed as arguments. Figure 4.1 shows part of a model definition written in the SID [30] input language.

The shape manipulation statements found in an input language can be divided into two types: transformations used for translating, rotating and scaling existing sub-models; and joining operators used for combining those sub-models in order to create more complicated shapes. The range of transformations available to the user varies between systems. It may be limited by restrictions placed on the user by the modelling system, the requirement for orthogonal placing in PADL-1 [26] (for

example). In order to be able to place a sub-model in an arbitrary position and orientation two transformations are required: the 'translate' transformation and the 'rotate' transformation. To allow the convenient manipulation of existing sub-models further transformations are desirable. A scaling transformation allows for the scaling of existing models and also mirroring.

The number of joining operators varies between systems. Some early systems had a single 'glue' operator, allowing two sub-models that have a common face to be joined. Most modern modelling systems offer the three set-theoretic operators *union*, *intersection* and *difference*.

Language constructs found in other computing languages (looping, testing and conditional constructs constructs for example) are also useful in model definition languages. For ease of use in defining shapes, a wide range of variable types for handling points, lines, planes, surfaces and models is required, as are mathematical and trigonometric functions for operating on them. An alternative approach to defining a complete language is to provide a range of variable type declarations and functions callable from an existing computer language, such as FORTRAN, C or Pascal. This approach is used in the input system to the TIPS [32] modelling system.

Graphical Input

Graphical input facilities are offered on a number of commercial modelling systems. A number of advantages are claimed for these in comparison with language input. Design engineers, architects and other users of systems are used to working

with graphical data representations. Hence they adapt more readily to graphical input techniques. Also many of these systems are offered as an upgrade to either two or three dimensional drafting packages and hence it is sensible to utilise the same user interface.

It would be ideal if it were possible for computers intelligently to recognise, to interpret, and to query sketches. However, this goal is far from being achieved at the moment. Programs [35], [36]and [37]which are able to address this task at the moment do so in the context of limited shape forms (plane-faced polyhedra), or place rigid restrictions on the style of the drawings used as input.

Those implementing practical systems have therefore tended to try to find 'half-way houses' which allow some graphical input, but in a form oriented to the requirements of the model's structure. One commercial system (Medusa from Cambridge Interactive Systems, now a division of Computervision Ltd.) allows the user to draw pointers between a number of views of an object to express the relationships between the views and to avoid the complexities of recognition. Less elaborate schemes have concentrated on the ability to input graphically simple sub-shapes which can then be combined by other (linguistic) means into more complex shapes. This may be achieved by instantiation of a range of commonly used component features, each only slightly more complex than the primitives in the system. Alternatively, facilities may be made available to design sub-shapes with particular limitations. 'Turned' parts are one example of this. Another is the 'perimeter object', a two-dimensional outline with a single thickness. These two facilities alone are widely applicable in the creation of models of mechanical

components.

The perimeter object was included by Braid in the early BUILD [38] system. It is now also available in the commercial system that has followed BUILD (Romulus [39]). Both these systems are B-rep modellers. The implementation of graphical input is more straightforward in this type of modeller, in the author's opinion, because the lines that form the sketch can be more or less directly translated into component edges in the model's data structure. It is more difficult to implement such input forms with set-theoretic modellers because in this case edges must be reinterpreted as the result of combining half-spaces or primitives.

Set-theoretic modellers do however, as outlined in Chapter 1, have several advantages however compared to their B-rep counterparts. Therefore it is worth developing graphical input for these modellers.

Since the algorithm described in this chapter was first reported [40] several other researchers have produced similar algorithms. Dobkin *et al* also report the work of Peterson [41] who has proved that it is always possible to generate a set-theoretic expression that contains a half-space for each edge that occurs only once in the expression. Tor and Middleditch [42] have developed a more complicated algorithm that is shown to have on average a near-linear time complexity.

Since the technique described in the next section was first reported, Peterson [43] has developed an algorithm capable of handling two-dimensional laminae, with boundaries constructed from straight lines and curved edge segments.

An Algorithm for Graphical Input of Set Theoretic Models

This section describes an algorithm for the graphical input of models to the set-theoretic modelling systems written at the University of Bath. The algorithm is capable of generating perimeter objects, perimeter objects with a draft angle and (faceted) shapes with rotational symmetry. It has been used in an interactive input system for the systems, as an input processor for geometric data from a computer aided part-programming package (SmartCAM [44]), and also used internally in the model generation stage of the toolpath verification system described in Chapter 6. The models created using the algorithm may be combined with other shapes within the language already used to feed the system.

The algorithm is designed to be capable of generating models of objects which may be described by closed curves in two dimensions (which is the reason that they are suitable for interactive graphical input). Hence the problem of identifying the half-spaces and their relationships may be reduced to that of finding a set of half-planes and a relationship between them which corresponds to the sketched profile. These half-planes may be expanded into half-spaces depending on whether the sketch represents a simple plate, a plate with a draft angle, or the cross-section through a turned part. There are a number of approaches to this identification problem. The one implemented is guaranteed to generate the minimum number of half-planes needed to represent the sketch (and hence the minimum number of half-spaces to represent the object). This reduces the complexity of the resulting model and helps to reduce computation times. Indeed the technique may well outperform a careless or hurried user using language input in the efficiency of

describing the object.

The algorithm works by decomposing the two-dimensional shape into a series of convex polygons. These are not a decomposition of the solid interior of the shape [45], which would add extra half-spaces to the description, but a relationship between both 'positive' and 'negative' regions. First the convex hull of the whole outline is found. Unless the outline is itself convex, there will be one or more regions of discrepancy between the hull and the original region. Further convex hulls are then found for each of these 'holes'. This may in turn leave undescribed regions, which will be 'solid' again. This process is repeated until there are no regions still to be processed. The shape can then be described as the outer hull, with the next layer of hulls removed, the layer after that added, and so on. (Each hull can be represented by the intersection of its constituent half-planes).

Using this method of arriving at a description would include artificially introduced half-planes. These are sides of the hulls that where not also parts of the original contour description, such as that marked with an * in figure 4.2.

If however, the description of the shape is constructed using the 'positive' and 'negative' convex hulls in the reverse order to that in which the layers were generated we find that, because the hulls are convex, the sides of the hulls that do not correspond to parts of the original definition may simply be omitted from the set-theoretic description. In effect, their places are taken by parts of the hulls of the opposite 'sign' from the next layer out. The resulting set-theoretic description uses *only* half-planes that correspond to portions of the original contour, and must therefore be minimal. Figure 4.2 shows a shape, its decomposition into a tree of hulls,

and the resulting set-theoretic algebraic description of the shape.

The problem of finding the convex hull of either the initial shape, or any of the nested sub-shapes, is essentially that of finding the convex hull of a set of points. A number of algorithms are available for this purpose [46]. The one selected is due to Jarvis [47]. It was chosen with a view to ease of implementation, especially the ability to deal with collinear points without too much special-case coding. The fact that this algorithm is not the most efficient is not considered to be a dominant factor, in view of the smallness of the point sets being processed.

The Interactive Input System

In the interactive input system that uses this shape decomposition algorithm the user sketches one or more outlines which are to become the cross-section of the object or part-object he wishes to describe on a graphics tablet. Each outline is described by a number of line segments. Because it will usually be necessary to associate exact dimensioning with the cross-section, the input is drawn on a grid. This has an initially regular pitch specified by the user, but individual grid lines may be perturbed to accommodate dimensions which are not multiples of the pitch. The pre-printed input sheet, which also has a menu of commands and a 'keyboard', is shown in figure 4.3. The layout is mirrored on a raster scan display. This can show the movement of the grid lines, and (by using separate pixel-planes for the grid and the input) allows editing of the shape to be performed without the confusion that would occur if the stylus trace on the input sheet were the only indication of the current situation. The lines drawn by the user are also straightened between

the tablet and the graphics display. When the user is satisfied with the shape he has created, he indicates that recognition should take place. The program links all the segments he has created into one or more closed figures. If some of these figures are nested inside one another, this is recognised and the nesting of these outlines is superimposed on that of the convex hulls. If any unclosed figures or crossing lines are discovered the program signals that an error has been made and returns the user to the figure input stage to correct these faults. If there are no errors, the decomposition algorithm is then invoked for each polygon in turn, and the entire structure of the cross-section is expressed in a form suitable for linking with language input, and passing to the solid modelling system.

If the user's input was to specify one or more flat plates, then it is simply necessary to add a zero coefficient for the third axis to the half-plane equations, and these become the description of a corresponding infinitely long 'extrusion'. The program asks the user to supply two planes perpendicular to the third axis to bound this extrusion.

Alternatively, because the algorithm establishes which side of each half-plane corresponds to the solid, this information may readily be used to modify the equation of the corresponding planar half-space. In particular, if each half-space is rotated through a constant angle, then a 'draft' angle may be applied to the shape. The rotation is easily applied by setting each half-space direction cosine corresponding to the direction which is normal to the drawing plane to a value with constant magnitude, with its sign depending on the side of the half-space which is solid. It should be noted that if the half-spaces are constructed in this manner, the

topology of the shape at the bounds of the 'extrusion' may be different from that drawn.

As a third alternative, the base line of the input grid may be interpreted as the centre-line of a 'turned' component. In this case, the centre line is used as a dummy side in the recognition process, and then discarded. Half-planes parallel to the centre line are interpreted as cylinder (which could be generated by a series of planar half-spaces arranged around the centre line and intersected together to form a faceted approximation). Lines at an angle to the centre line are interpreted as cones (again faceted if this is required by the modelling system). Only lines perpendicular to the centre line are interpreted as single planar half-spaces. If faceted models are to be generated then the distance of the input points from the centre-line determines the number of facets on each curved surface. This is under the overall control of the user, who specifies the degree of conformance he requires.

An Example

Figure 4.3 shows a prepared input sheet with a sketch describing a plastic vice jaw. The first stage of the input procedure is to specify the plotting grid. The grid and the associated coordinate values are displayed on the screen. At this point the only part of the tablet that is 'live' is the menu command area. The user indicates that he is about to sketch the outline of a component by making a mark in the 'Draw' box. He then sketches the outline, making certain that points with different dimensions in a coordinate are drawn on grid lines. If any errors are detected (such as lines with the same start and end point or lines outside the plotting region) this is

indicated to the user and the line is discarded. Parts of the outline can be changed by using the 'Erase' command and then redrawing the required modifications.

As can be seen in figure 4.4, some of the grid lines have been moved in order to dimension the component correctly. This is achieved by pointing first to the 'Position' command box, and then to the line to be moved by making a mark in one of the boxes adjacent to the lower or left hand edges of the plotting grid. The new value for the grid line is then entered using the numeric keypad on the input sheet. If this value is acceptable (it does not overlap adjacent lines), the old grid line is replaced by the new line and this is reflected on the display. When the user has completed the input process he uses the 'Finish' command to start the recognition procedure. The 'Wait' command may be used at any time to allow the user to write comments on the input sheet (for example the name of the component). A hardcopy of the display screen, together with the input sheet provides a permanent record of the dimensioning of the component. Figure 4.5 shows a view of the component as produced by the modeller. The transverse slot in the base of the component was entered on a second sheet and then 'differenced' from the component in the language form.

Building a Model from Outlines from a Computer Aided Part Programming Package

Computer Aided Design and Computer Aided Part Programming packages are in widespread use for generating toolpaths for numerically controlled machine tools. The algorithm described above has been used in a input processor that takes

graphical data, in the form of outline contours from such a system, and generates a solid model from such data. An example of such use is shown in figures 4.6 and 4.7. Figure 4.6 shows the outline of a component in the SmartCAM system. Figure 4.7 shows the component as modelled by a solid modelling system, the model having been created from the graphical outline data.

It may be noted that this automatic generation of a model from graphical input could be used as the basis of a toolpath checking system based on 'subtracting' the swept volume model from the model generated from the outline. (This is *not* the approach used in the verification system described in this thesis.) If this approach were to be taken, then it would be of limited use since any curves in the outlines imported from the system would need to be faceted. It would then be difficult to avoid spurious discrepancies being detected between the facetted outline model and that generated by the swept tool volume (for a more detailed description of the process, refer to Chapter 5).

```
Sets {object}

FUNCTION box(minpt: Point, maxpt: Point): Set
; This generates a box aligned with its two leading diagonal corners at minpt and maxpt.

Sets {minxf, minyf, minzf, maxxf, maxyf, maxzf}
Points {x_vec, y_vec, z_vec}
{; Three coordinate unit vectors
                    x_vec := pt(1,0,0)
                    y_vec := pt(0,1,0)
                    z_vec := pt(0,0,1)
; Six box faces
                    minxf := space(-x_vec,minpt)
                    minyf := space(-y_vec,minpt)
                    minzf := space(-z_vec,minpt)
                    maxxf := space(x_vec,maxpt)
                    maxyf := space(y_vec,maxpt)
                    maxzf := space(z_vec,maxpt)
; Box is the intersection of the faces
                    RETURN(minxf & minyf & minzf & maxxf & maxyf & maxzf)
}

FUNCTION sphere(centre: Point, radius: Real): Set
; This generates a sphere centred at centre, with radius radius.

Sets {x_p, y_p, z_p}
{; Create three mutually perpendicular planes through the centre
                    x_p = space(centre + pt(1,0,0), centre)
                    y_p = space(centre + pt(0,1,0), centre)
                    z_p = space(centre + pt(0,0,1), centre)
; Build the sphere and return it
                    RETURN(x_p*x_p + y_p*y_p + z_p*z_p - radius*radius)
}

; ---------- Main program.  Build a sphere with a box through it.
{
                    object = colour(sphere(pt(0,0,0),1),1) | colour(box(pt(-0.5,-0.5,-1.5),pt(0.5,0.5,1.5)),2)
                    write("sphere_box",object)
}
```

Figure 4.1 An example of language input

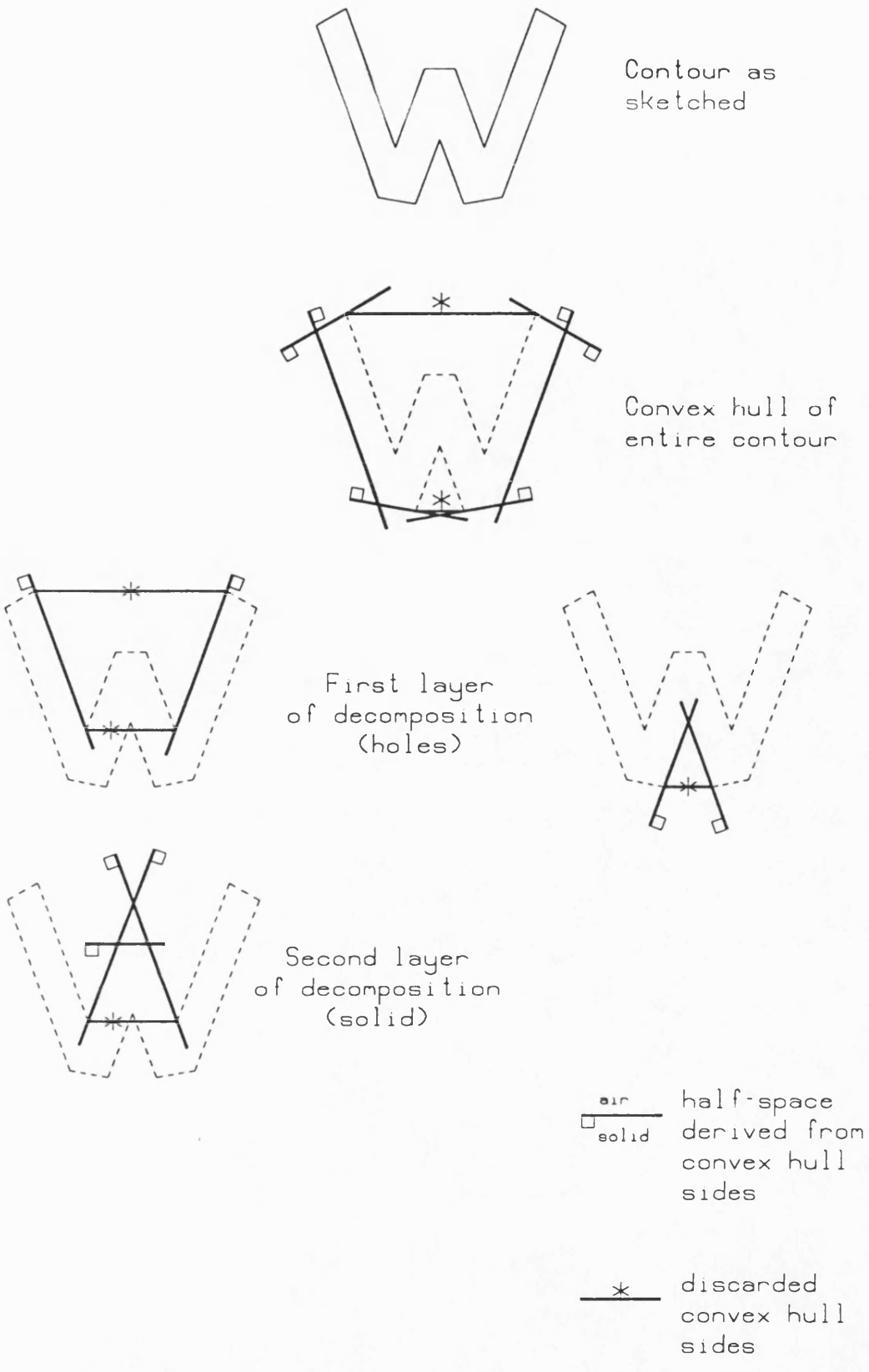

Figure 4.2 A graphical input algorithm

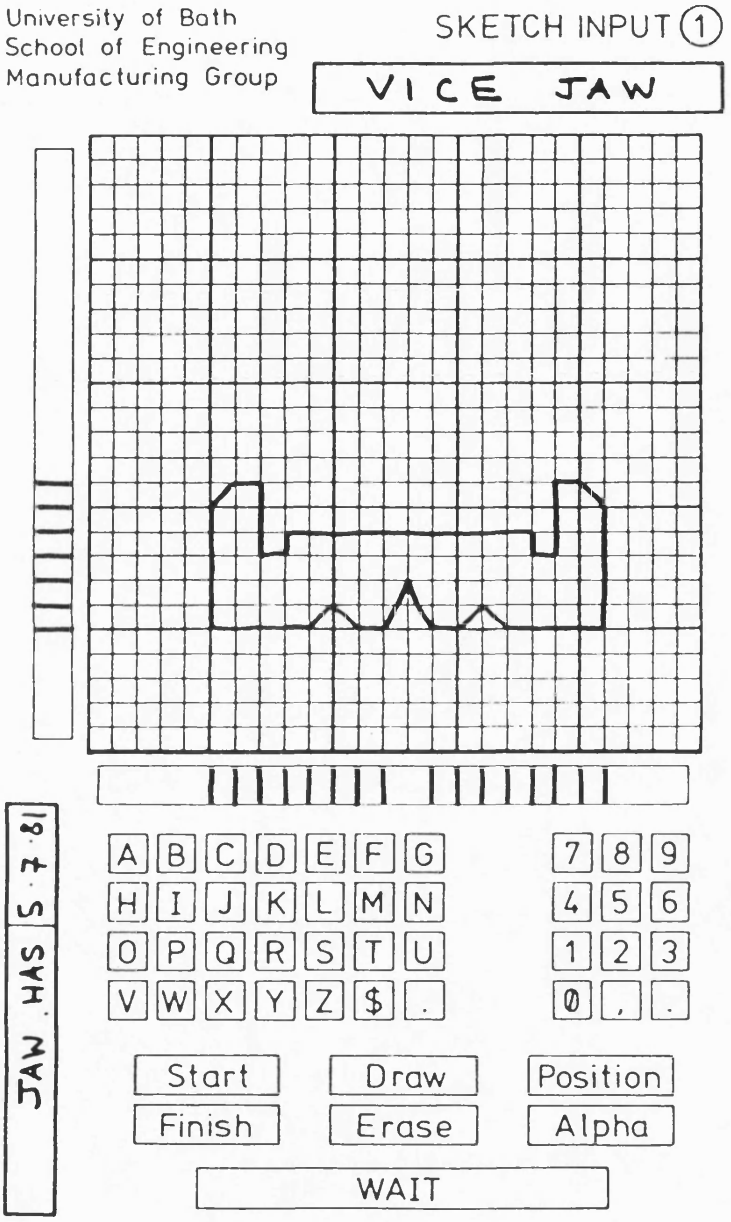

Figure 4.3 Input Sheet for Graphical Input System

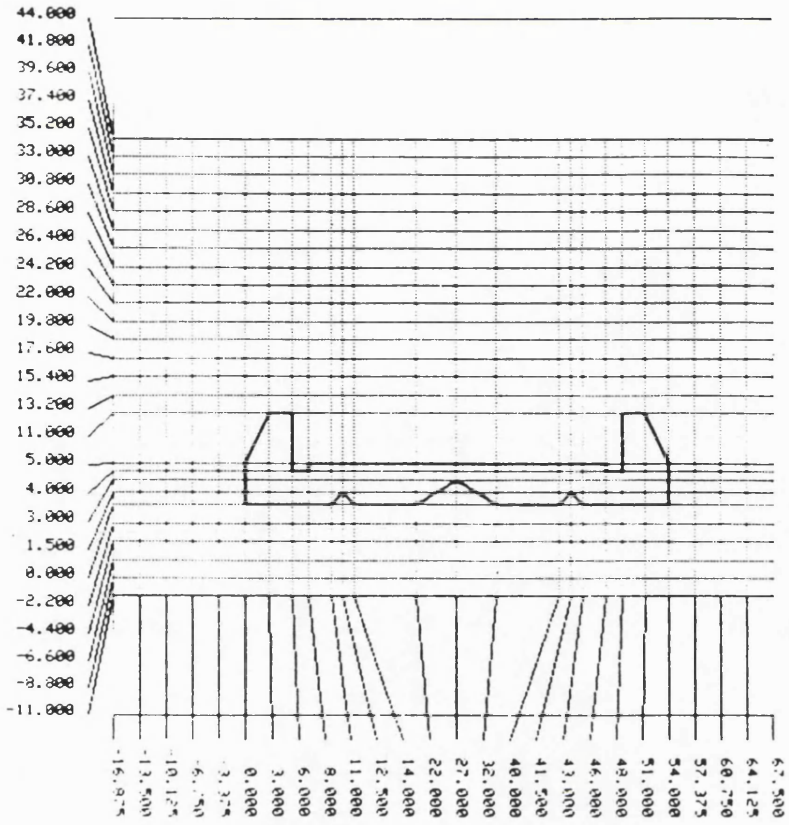

Figure 4.4 Screen Display for Graphical Input System

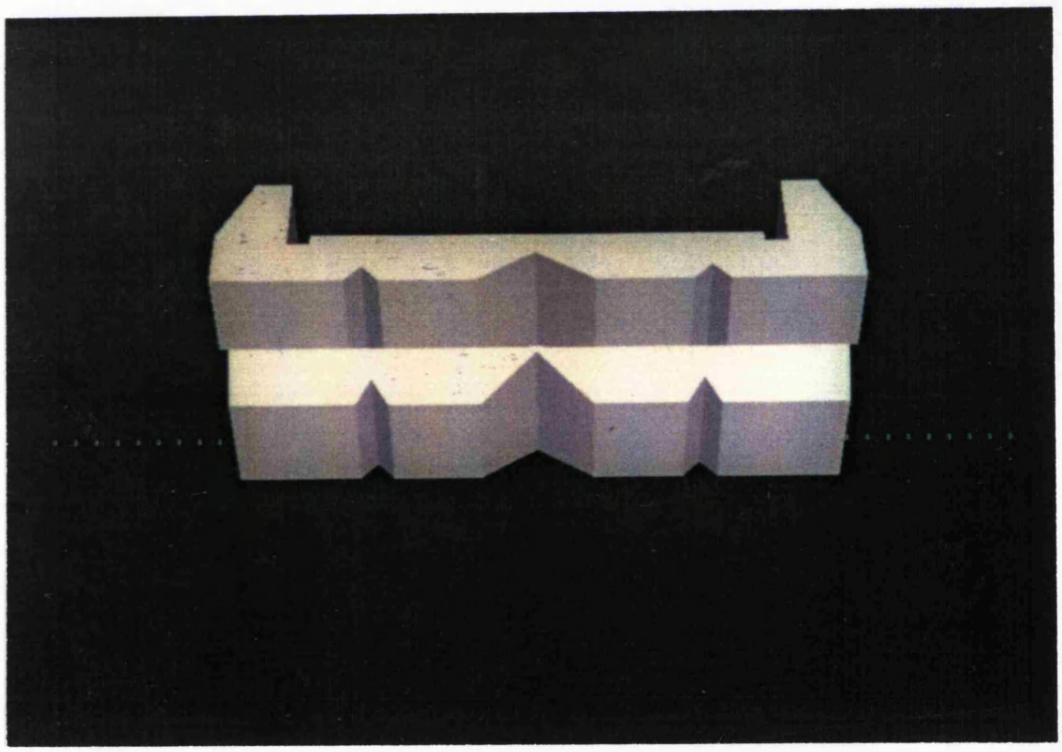

Figure 4.5 The shape as modelled using 'DODO'

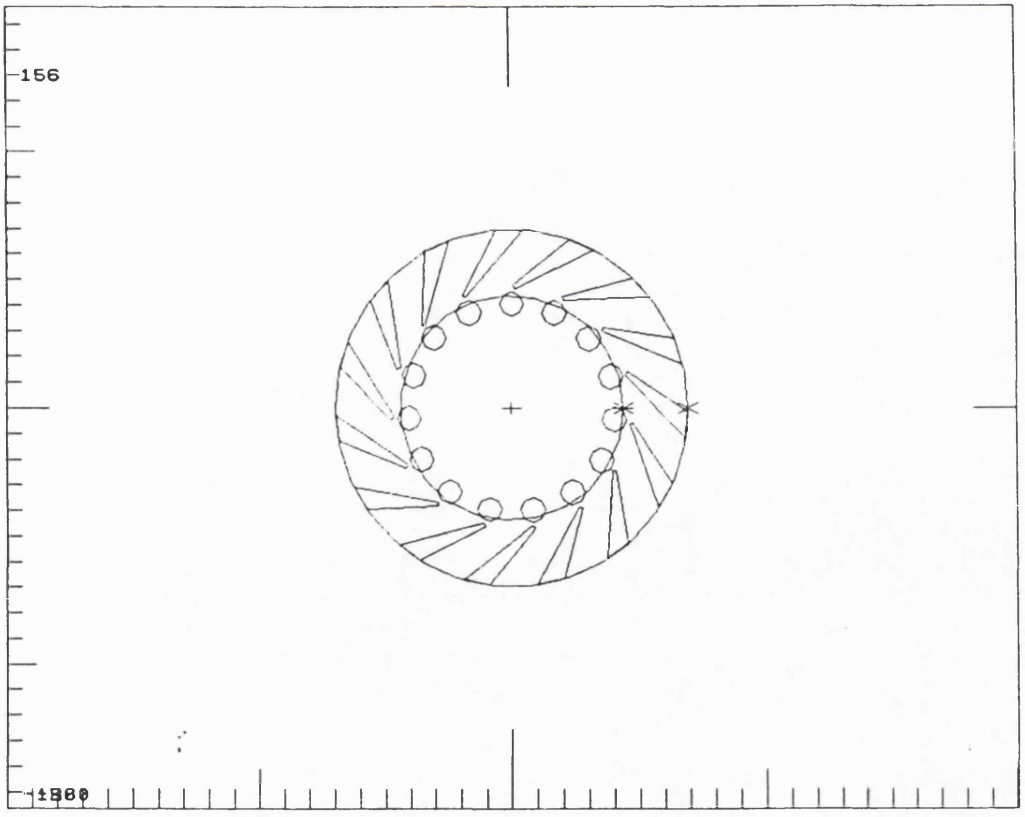

Figure 4.6 A 'screen-dump' from SmartCAM

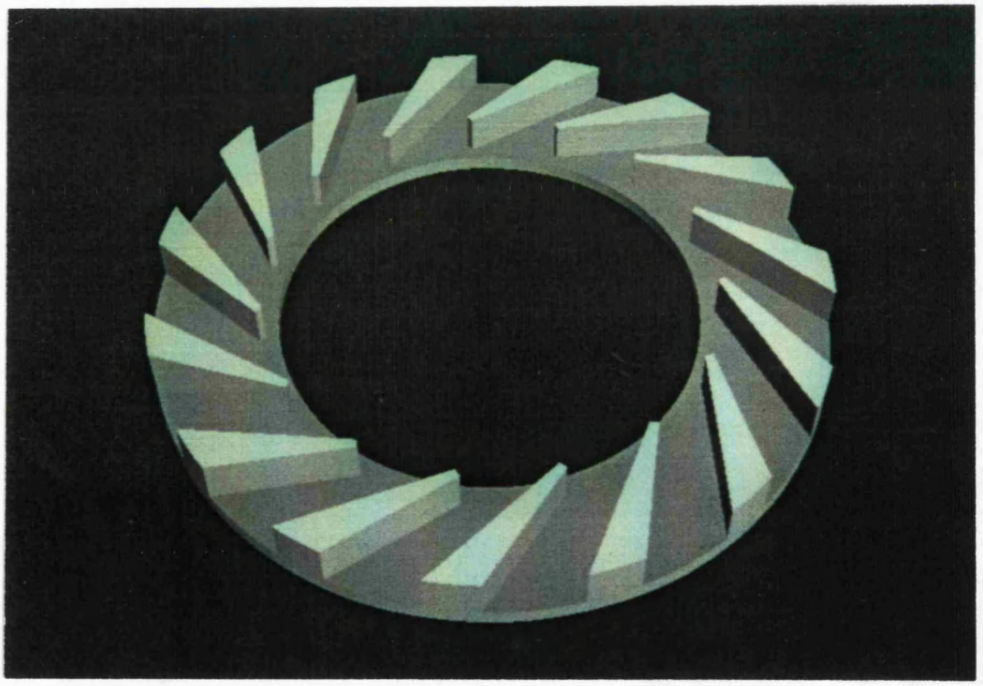

Figure 4.7 The shape as modelled using 'DODO'

CHAPTER 5

Toolpath Verification

Toolpath Generation

Since the introduction of numerically controlled (NC) machine tools in the mid 1950s their use in engineering industry has increased rapidly, and at present they are in widespread use. Many different machine tools may be fitted with numerical control: most commonly lathes and vertical milling machines. Some of the first numerically controlled milling machines were capable only of moving the tool such that its motion was orthogonal to one of the machine axes. Other machine tools were capable of contour cutting (ie moving more than one machine axis simultaneously), but many were restricted to simple point to point (linear) movements because of the high price of control systems for contouring.

With these simple machines, programming was easily performed manually. The limited tool movements meant that separate toolpath verification was not required. Also, it was often the case that programming was done on the machine-tool itself, by manually instructing the machine to move through the required sequence of operations, whilst simultaneously recording the tool movements on paper (or magnetic) tape. In this case, separate toolpath verification is not really necessary.

The next generation of NC machine tools were capable of more complicated tool movements. These included 2 or 3 axis linear interpolation and 2 axis circular

interpolation. Also automatic toolchanging mechanisms were added allowing the cutting tool to be changed under program control. Components designed for manufacture on such machines often have edges described by line and arc segments that lie at arbitrary angles to each other and to the coordinate system. Fillet radii that lie tangentially to these lines and arcs are also frequently specified. The manual programming of toolpaths for the components involves many geometric calculations (often calculating the positions of the centers of arcs that lie tangential to other arcs and lines) and is extremely time consuming and prone to errors.

In the 1970s several computer assisted part-programming packages were developed. Typical of these, and still in widespread use, is G.N.C. (Graphical Numerical Control) written at the CAD Centre [48] . As originally written it allows the user to generate toolpaths for machining two-and-a-half dimensional components on vertical milling machines. (Two-and-a-half dimensional components are those which may be have an arbitrary outline in two dimensions, but only step changes in the third and leave no undercuts.)

As with other computer assisted part programming packages, such as Pafec's DOGS-NC and Point Control's SmartCAM [44] , that are written for generating two or two-and-a-half dimensional toolpaths, the generation process starts with the user interactively defining the geometry of the outlines of the component in terms of unbounded lines and circles. These may then be joined together to form continuous outlines. Cutter paths may then be defined based on these outlines by 'driving' a specified cutter along the outlines from between specified start and end points, and with a specified offset direction. If a closed outline is defined, a

roughing operation may be automatically performed, generating a spiral, or linear (zig-zag) toolpath that removes material from the region enclosed by the outline. These cutter motions may then be joined together with other operations, such as hole drilling, to form a complete toolpath.

Other packages such as Polysurf and Duct will generate toolpaths for three dimensional curved surfaces.

One problem with all these packages is that - although they contain a description of the geometry of the individual parts of the surface of the component - they are not, in general, capable of detecting whether a given tool movement will cut into other parts of the component. This results from both the nature of their internal representation of the component, and the method of the packages use. Often the representation scheme is not capable of answering queries such as whether a point lies inside or outside the component. This may make it difficult to detect intersections between the toolpath and parts of the component. When using these packages, the user will specify a surface that is to be machined, together with a number of 'check' surfaces that contain the motion of the cutter. Other surfaces of the component may be ignored, so if the tool cuts these surfaces, this may not be detected.

The Complexity of Toolpaths

The increase in sophistication of both computer-assisted part-programming systems and N.C. machine tools has led to an increase in the complexity of toolpaths. There are several factors that contribute to the complexity of such a toolpath.

There may be a large number of separate, possibly simple, tool movements. This is often the case where some form of three dimensional 'sculptured' surface is to be machined using a large number of small linear (or circular arc) cuts. Alternatively the tool motions themselves may be geometrically complex. Many modern milling machines, for example, are capable of performing helical cutter motions. Also, there are now many machine tools that are capable of performing 4 or 5 axis motions. The shapes of the tools themselves may also be complicated; in milling for example a barrel-shaped cutter with corner radii might be used. It should be noted that the toolpath for even a simple shape may be complicated since several passes may be required to remove material in roughing operations followed by one or two finishing cuts. Also, if the surface geometry of the component is not well suited to the range of cutter shapes and machine movements (or such movements cannot be calculated) then a large number of short cuts may be required.

The Need for Verification

It is often desirable that these complicated toolpaths be checked before the machining of an actual component is attempted. There are several reasons for this. Firstly, if the toolpath is not correct then damage may be caused to the machine tool, vice, fixtures and cutting tool. Incorrect machining can often mean that the component blank has to be scrapped or requires expensive reworking. This is especially costly if the blank is of high value either because of its size and/or material, or because it has already undergone an expensive pre-machining process. Manufacturing incorrect components also wastes machine time, especially if the

errors are not detected until after the component (or maybe several components) have been machined. An N.E.L. report published in 1975 [49] surveying NC2+2 machining is the USA stated that up to four reworks were required to correct errors in toolpaths.

There are a number of potential sources of error in toolpaths:

- Human error in specifying coordinate information, or in using the computer-aided part-programming software.
- Errors in the computer aided part-programming software.
- Errors in the post-processing software.
- Incorrect tools loaded into the machine-tool tool magazine, or incorrect fixture or tool offsets specified.
- The toolpath is geometrically correct, but an unwise choice of the order of machining, poor selection of feedrates, spindle speeds or depths of cut results in errors.
- Parts of the tool holders or machine spindle assembly collide with the component, fixturing or the machine bed.

Ideally a verification system should be capable of detecting errors from all of these sources.

Requirements of the Verification Process

Before looking at the various possible methods of performing toolpath verification it is worth considering what is required of the toolpath verification process.

The most important requirement is that it should check that the shape of the component resulting from the cutting operation is that which is required.

This means checking both in terms of its overall shape and also detailed dimensional measurements. Verification should check for potential collisions between the cutting tool and those parts of the machine tool that move with it, and the component to be cut, any fixtures, and parts of the machine. The are also a number of technological factors that could be checked. These include checking that the speeds and feed rates are suitable for the depth of cut and the material to be used, and also that the direction of travel of the tool is consistent with its function (that a twist drill is not to be used to cut a slot for example).

Also, errors that generate the correct shape, but would also result in collisions between the cutters or parts of the machine-tool should be detected.

Since errors may occur either before or after the post-processor, any system should ideally be capable of use with output directly from the toolpath generation stage, or after post-processing.

As far as is possible, errors caused by wrong tool loading, or wrong fixturing should also be detected.

Methods of Verification

Toolpath verification can be performed in a number of ways. The simplest (and probably most commonly used) methods requires only the machine tool that is to be used to cut the component. The most basic of these is to set the machine tool to perform all movements at a (predefined) slow feedrate. Alternatively it

could be set either to ignore z-axis movements or to machine the component blank with the z-axis coordinates offset. All tool movements should now take place above the surface of the component blank. The toolpath may now be run on the machine, possibly in 'single block' mode and/or with a reduced feedrate under the supervision of the operator. There are a number of obvious drawbacks with these simple methods. The technique is only suitable for checking for any large errors in the toolpath since there is no record of the path followed by the tool. It is only applicable for checking toolpaths that involve mainly one or two dimensional cuts. Toolpaths for machining 'sculptured surface' components may consist of many tens of thousands of cuts, hence at reduced feedrates, the machining times for verification purposes may become extremely large.

Another traditional method of toolpath verification is to use a substitute material. The component blank is replaced by a similar shaped blank made of a material such as foam, wood or wax. The toolpath is checked by running it on the machine tool.

Due to the good machinability of the substitute material the feedrates can be increased, resulting in a verification time that is less than the actual machining time. As with the previous technique, no extra capital cost is involved. Substitute materials that are easy to machine, such as wax and foam, often display dimensional instability. This means that it is not possible to check the dimensions of the component accurately. Some substitute materials are also very messy to use. The lack of mechanical strength of materials such as wax may also be a problem if thin sections are to be machined; in many cases a blank that is larger than the actual

component blank is required to provide sufficient mechanical strength and stability. Also, errors such as excessive cutting depths, or incorrect spindle speeds will not be detected due to the much reduced cutting forces generated when machining the substitute material.

In any technique that uses the machine tool, large errors in the program may result in damage to the machine tool. Also, after machining, the resulting shape displays the effect of all the machining steps. If there is an error in the shape then it may not be obvious which part of the toolpath caused the error and little information is gained. For these two reasons the verification process must be watched by the programmer or machine operator. Clearly, whilst being used for toolpath verification, the machine tool is not performing useful production tasks. Proving times of up to six hours per hour of machining time are not unknown.

Computer Verification

Early attempts at computer verification of N.C. toolpaths were hampered both by the high cost of computers of sufficient power, and also the unavailability of suitable graphics output devices.

The fall in the cost of both computers with sufficient power and of graphics output devices has led to the development of computer based toolpath verification systems. The simplest programs display the path followed by the tool relative to the workpiece. This may be plotted as a line on a graphics screen or plotter, often in two or more views, usually in either orthogonal or isometric projection. Differing line types, and/or colours may be used to differentiate between rapid and

feed-rate movements.

Simple centre line plotting is often available as part of graphical computer assisted toolpath generation packages. Many modern machine tool controllers are capable of plotting simple orthogonal or isometric views the the tool motion, either whilst the tool is moving, or, more usefully for toolpath verification, whilst keeping the tool stationary.

There are several disadvantages to this simple approach. If the toolpath includes three-axis movements, or a large number of cuts, then the display is difficult to interpret. The plots are often far more complicated than the surfaces that they represent. For example, a typical pocket roughing operation will move the tool back and forth within the confines of the pocket. This will result in a number of lines in the plot, but only one surface in the component. The apparent advantages offered by having more than one view are lessened since it becomes difficult to correlate the cuts between views. Also, unless block numbers are printed alongside the lines, which further complicates the display, it is not easy to relate the plot to toolpath that generated it.

The major disadvantage of the centre-line plot is that it only displays the path followed by a point on the tool. In order to visualise the surface created by the toolpath the user has to 'add-on' the shape of the tool. The situation is further complicated if more than one cutting tool is used.

A drawing of a simple component, drawn on a CAD system is shown in figure 5.1. Figure 5.2 shows a tool centre-line plot for the component from the

SmartCAM part-programming system; even for this relatively simple two-and-a-half dimensional shape, it is not easy to interpret the plot. The component was cut in a substitute material, a picture of which is shown in figure 5.3. In order to provide sufficient support for the foam during machining, large amounts of excess material are left surrounding the limits of the actual component blank. The actual component machined from aluminium is shown in figure 5.4. Figures 5.5 and 5.6 show a drawing of a more complicated component, and a centre-line plot of the machining sequence for the upper side of a set of six components.

Several other, more complicated, graphical approaches have been tried. These all attempt to model in some way the volume swept by the cutting tool. The technique adopted by McGoldrick and Gibson [50] was to draw orthogonal views and sections of the outline of the cutter path. This was easier to interpret than the centre-line plot and overcame some of its problems. Their system could handle two-and-a-half dimensional toolpaths. Any other limitations imposed on the toolpaths are not given in their paper. It is not clear how the technique could be extended to cope with more complicated tool movements. It is not possible to extract accurate dimensional information from the views.

An alternative technique was developed by Anderson [51] to detect certain types of errors in NC toolpaths for vertical milling machines. The technique will detect fouling between the toolholder and the (partially machined) component. On detecting a tool motion that would result in such a collision it will attempt to generate a corrected tool movement. The toolpaths must not contain any undercuts. The algorithm creates a two dimensional array of heights for points on an (x,y)

grid with each element initialised to contain the z height of the initial billet. As each tool movement is processed the z heights for each element that the tool passes over are updated to the height of the tool base. Intersections between the toolholder and the current billet surface are detected by checking the height of elements that the toolholder passes over. As Anderson states, the resolution of the system is limited, and he suggests a grid with an element spacing of 10 to 20 percent of the cutter diameter. The system is limited to cylindrical cutters, and assumes that the toolholder (and any relevant parts of the machine head) may be represented by a number of cylinders concentric with the cutting tool.

Chappel [52] describes a technique that represents the material removed during a milling operation using vectors. A mesh of points are defined on the surface of the required shape. Vectors are generated that pass through these point and lie normal to the surface. These vectors are extended both into and away from the surface, bounded by other parts of the required surface and the surface of the original billet. The cutting tool is modeled as a cylinder. In order to simulate machining the cylinder is instantiated at discrete positions along the toolpath and the vectors that intersect the cylinder are clipped against it. Collisions between the toolholder and machine head (modeled again by a number of concentric cylinders) are detected by checking for intersections between each of these cylinders and the vectors. The checking is based on a general vector-cylinder test and is thus not restricted to toolpaths in two or three axis.

The main limitations of the approach are that it only checks the path at discrete locations, rather than in its entirety. Also, as with the previous technique,

it it limited to cylindrical cutters and a cylindrical approximation to the toolholder and machine head and spindle assembly. The feedback to the user of the system is also limited. All the examples given are for relatively simple toolpaths.

The approach taken by van Hook [53] is based on a hardware Z-buffer. Continuous-tone pictures of the (partially machined) component are generated on a special raster-screen display. For each pixel in the image a linked-list is kept of the distance to each surface of the component blank that lies behind that pixel. For each tool position that is to be checked, the set of pixels corresponding to projected image of the tool is found. The distance to the near and far sides of the tool for this pixel are calculated and the linked-list is updated. If this results in a different surface becoming the 'front' surface then the screen pixel is updated. The algorithm is implemented in microcode and can display movements in real-time. As in the previous two techniques, the toolpath is checked by instantiating the cutter at discrete locations, rather than checking the volume that is swept by the cutter as it moves.

All of these techniques suffer from the limitations that result from the choice of data-structure used to represent the component and/or material removed by the tool. One effect of this is the absence of any facilities to allow the user to obtain non-pictorial information. Another is that several of the techniques are limited such that the tool position is checked only at discrete locations, which may lead to some toolpath errors not being detected, and apparent errors being introduced. The lack of facilities to ask queries of the system (for example by using a cursor to point at features in the picture) increases the effect of these limitations.

Toolpath Verification by Solid Modelling

In order to represent the geometric component blank, the cutter, and the machine-tool in a complete manner a data-structure is required that will represent the geometry of these in three dimensional space. One method of representing such volumes is to use geometric modelling techniques.

The simplest way of using solid modelling techniques to perform toolpath verification is as follows. A model is constructed of the volume swept by each individual cutter movement. These are then unioned together, resulting in a model of the total volume swept by the cutter. A separate model is constructed of the component blank. The toolpath model is then differenced from this model to give a model of the resulting component. From this model views of the component may be generated. These views are then examined to see if the component, and hence the toolpath, is correct.

If models of parts of the machine tool, clamps and so forth are unioned with the component blank model then collisions between the tool and these can be visually detected in the final model.

A verification system [54] of this type based on the TIPS [32] modelling system has been developed by Fridshal *et al.* This system used the user-definable surface features of TIPS to allows 4 and 5 axis tool motions to processed. In addition to generating shaded pictures of the model resulting from subtracting the toolpath from the model of the component blank, the system also allows the user to ask queries of the modeller by pointing at a screen displaying the picture.

In a report issued by the Production Automation Project at Rochester University [26] Hunt and Voelcker describe an experimental verification system based on the PADL-1 geometric modeller. This system requires the user to create a solid model of the component that is to be machined. This is then compared to the model of the component that results from processing the toolpath and any discrepancies are detected. If discrepancies do exist, then they are assumed to be errors in the toolpath. The report also discusses a number of issues concerned with implementing an incremental verification system, based on a general solid modeller, that will perform verification on a block-by-block basis.

One of the advantages of the system is that the toolpath may by automatically checked once the model of the required component is defined, although this strategy does have several potential problems. Firstly, the user has to construct the model of the required component (in practice such a model may be available from the CAD system used to design it). Secondly, if the component model does not correspond exactly to that resulting from machining, then spurious errors may be reported. Hence, even regions that are not to be machined have to be modelled identically in both the model of the original billet and in the component model. The comparison of the two models requires that a 'same object test' or 'null object test' be performed on the two models. Since the two models are constructed entirely differently, and discrepancies between them may be very small, there is a large scope for numerical problems occurring.

One of the conclusions reached by Hunt and Voelcker is that for an incremental verification scheme, a pure set-theoretic modelling scheme is computationally

less suitable than either a pure B-rep or a dual representation scheme due to its inability to use intermediate results.

Problems with using Solid Modelling for Toolpath Verification

In practice there are a number of considerations that make many 'standard' geometric modelling systems ill suited for performing toolpath verification. The choice of representation scheme, the internal data-structure, and of the form of graphical output all need consideration when choosing a modelling strategy.

The models generated from the toolpath tend to be very large, much larger than a simple model of the resulting component. This is because in order to generate even a simple surface a large number of cuts may be required. The non-linear response of many geometric modelling techniques to model complexity leads to long processing times.

A second potential problem is the wide range of differing surface and (especially) edge geometries that are required to be modeled. The movement of a cylindrical cutter in a straight line not orthogonal to any of its axes, for example, requires the modelling system to be capable of representing planar, cylindrical and elliptical prism surfaces. If a boundary-file is to be generated then the intersection curves of all these surfaces must be representable. Fridshal [54] reports that modifications were needed to the TIPS modelling system to make it suitable for toolpath verification. The changes required were firstly to extend the range of surfaces that could be modeled (employing the user-definable surface features of TIPS), and also to remove limitations on the distribution of set-theoretic operators

in the set-theoretic model.

As explained in Chapter 2, line drawings are not particularly suitable for displaying the curved surfaces that often occur in models generated by toolpaths. If they are used, hidden line elimination will be essential to allow the user to understand the images, owing to the large number of lines that would otherwise be displayed. Tools with rounded ends or corners generate surfaces with few sharp edges in the model. Therefore it is likely that additional lines will be required to portray the curved nature of the surfaces.

As with all graphical verification techniques, the graphical output gives only overall geometric information. There is also a requirement to be able to examine in detail the model resulting from the verification process. This implies that the user must be able to make queries of the modelling system, preferably in an interactive manner with graphical feedback. The range of such queries should be such that detailed geometric data may be extracted from the model (either the model of the component resulting from the tool sweeping operation, or that resulting from a comparison of such a model with the model of the 'ideal' component) so that potential errors may be checked.

If errors are found (either by being detected automatically by the system, or during interrogation by the user) then it must be possible to relate these errors back to the toolpath generation process so that the cause of the errors may be discovered. This will typically mean that the block number or program line number for the erroneous tool movement will be required.

If the verification process is based on the comparison of a model generated from the toolpath swept-volume and a model of the required component then two further problems have to be overcome. The first occurs when sculptured surfaces are to be machined using a ball-nosed cutter. In this case, the designed surface may be a smooth surface whereas the corresponding machined surface may consist of a number of separate surfaces, each generated by a small tool movement. (The surfaces would normally be smoothed after machining by hand-finishing.) The verification system must therefore be able to distinguish between these discrepancies and genuine errors. The second problem occurs when surfaces to be machined have tolerances specified. The surfaces resulting from machining may differ from the designed surfaces, whilst remaining within the specified tolerances. In order to overcome these problems the 'same object test' or 'null object test' would have to take account of surface tolerancing. Clearly a modelling system using facetted primitives is unsuitable for this approach.

The Approach taken in this Project

Despite these problems, the toolpath verification system described in the following chapters uses solid modelling techniques. A set-theoretic modelling system is used, using spatial division to reduce the effect of the size of the models generated from the toolpaths. The approach adopted in the toolpath verification system described in the following chapters is as follows (see also figure 5.7):

1 A geometric model is created of the volume swept by the cutting tools as they follow the toolpath. This model is then combined with a model of the com-

ponent blank, together with models of any clamps etc. These processes are described in the Chapter 6.

2 The combined model is then spatially divided for more efficient evaluation. This process is described in Chapter 7.

3 A number of images of the divided model that represents the result of the machining operation are generated, using viewing parameters defined by the user. At this stage, any gross errors in the component may be seen. This stage, and the next are described in Chapter 8.

4 Facilities are provided for the user to interrogate the model, using a cursor to point to features in the images. Thus detailed geometric information may be extracted from the model.

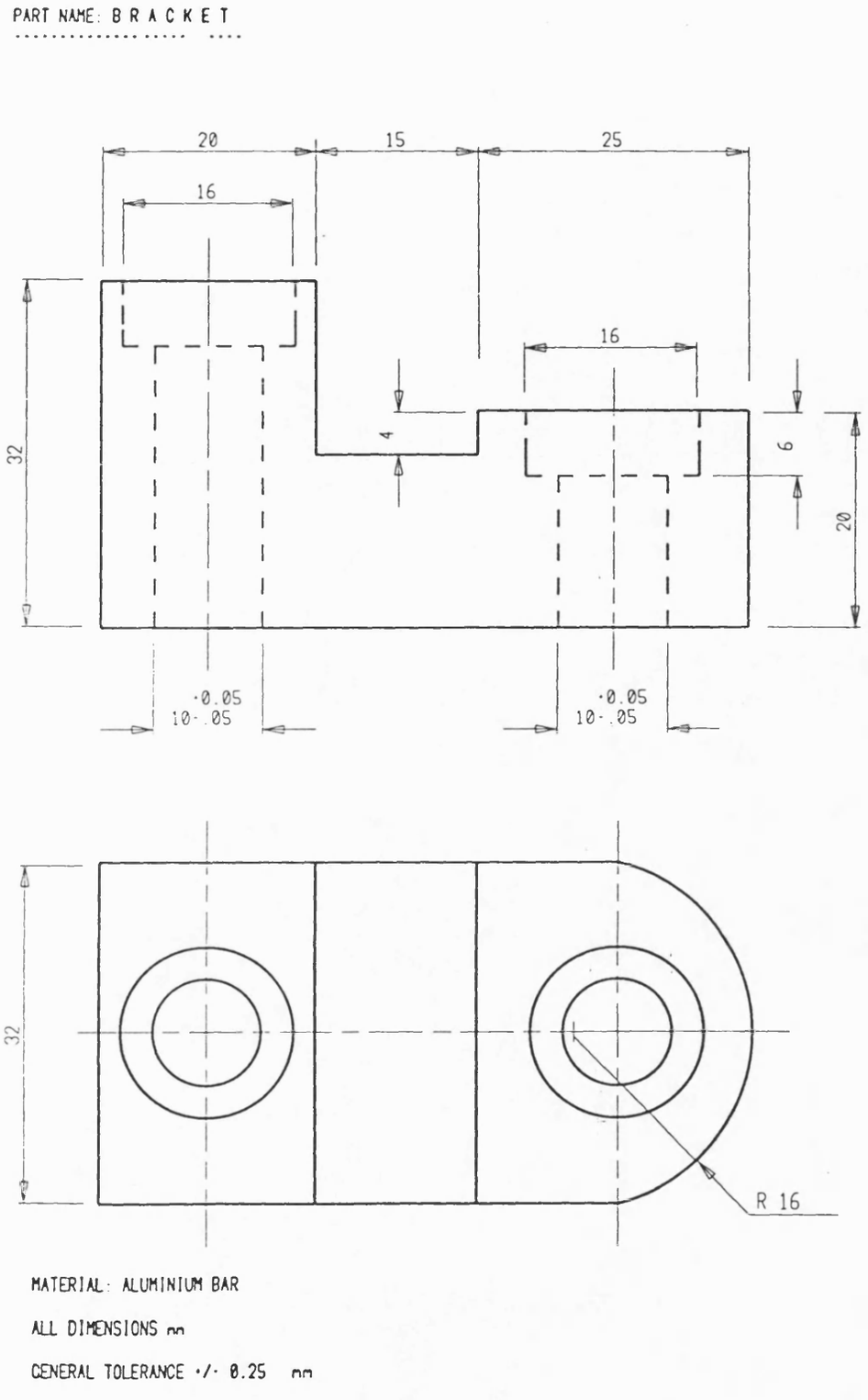

Figure 5.1 A Drawing of a Sample Component

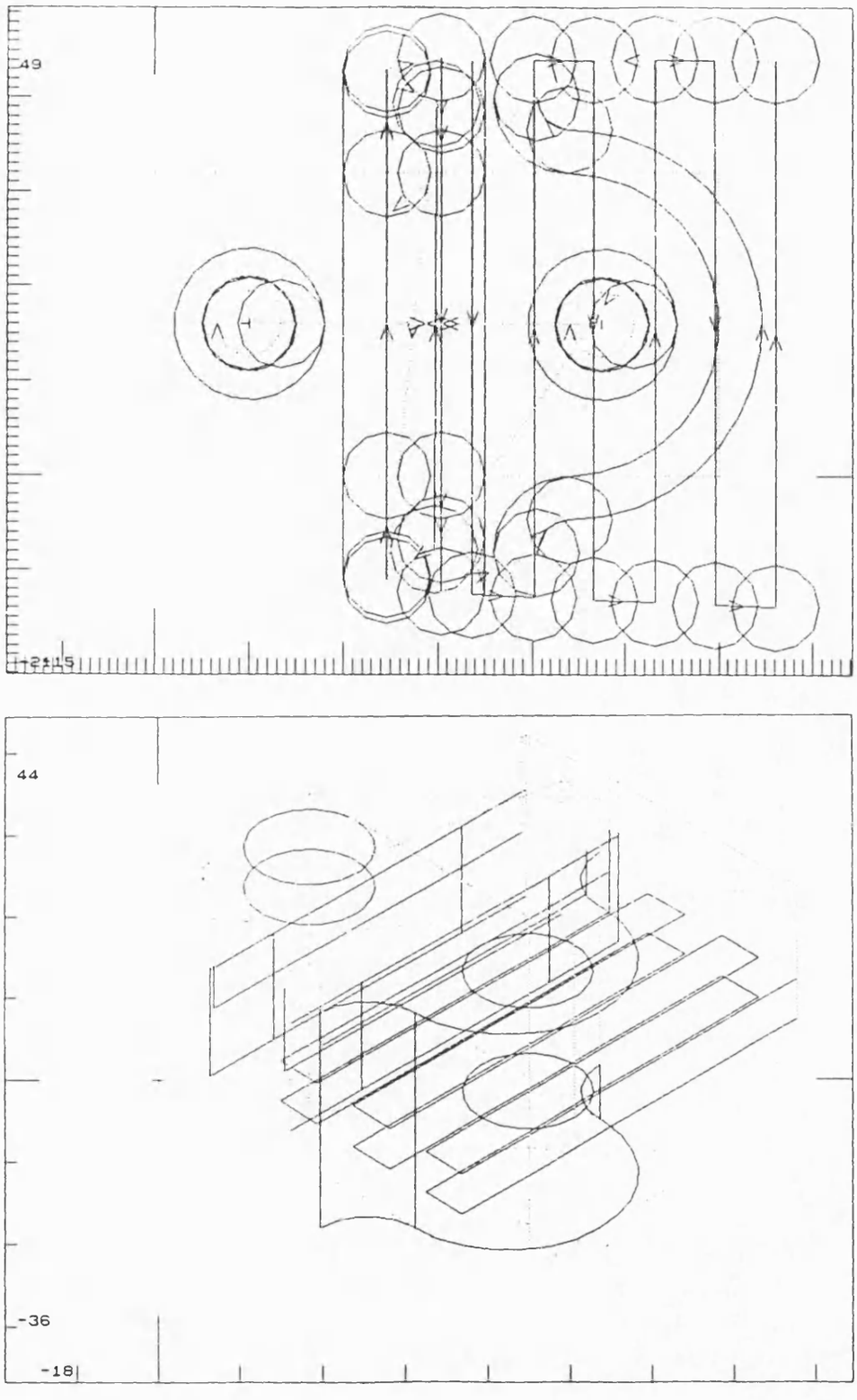

Figure 5.2 Vector-plot of toolpath centre-line

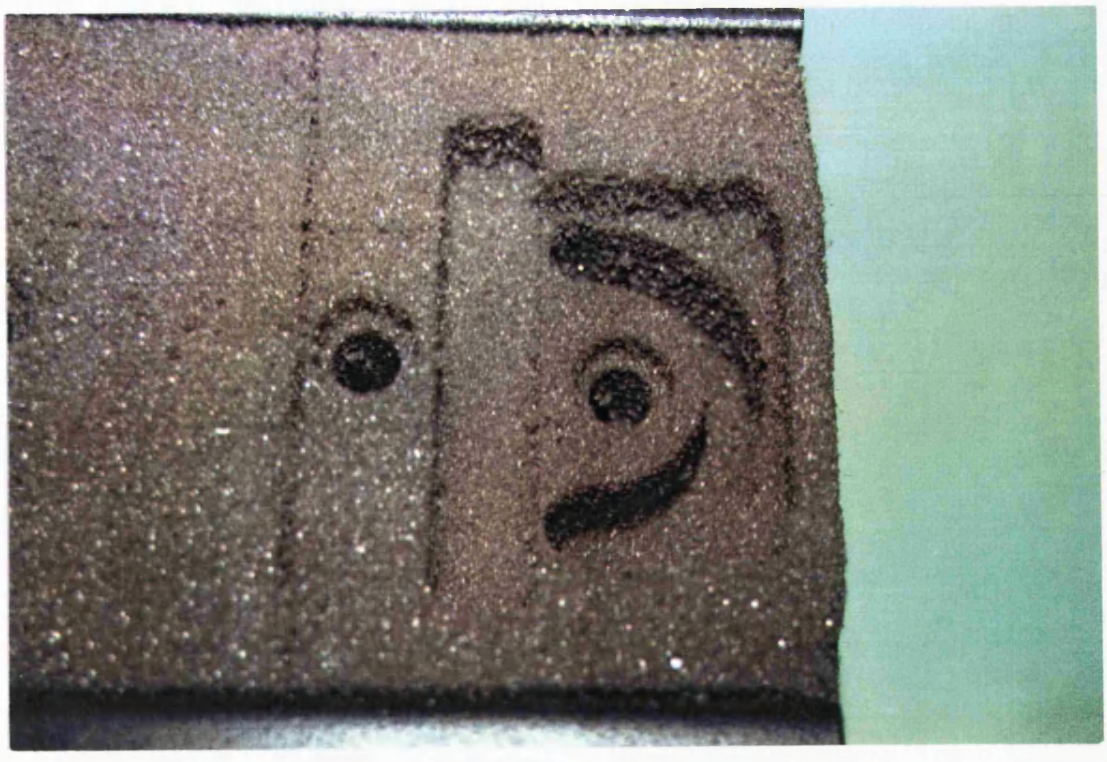

Figure 5.3 The Component cut in Foam

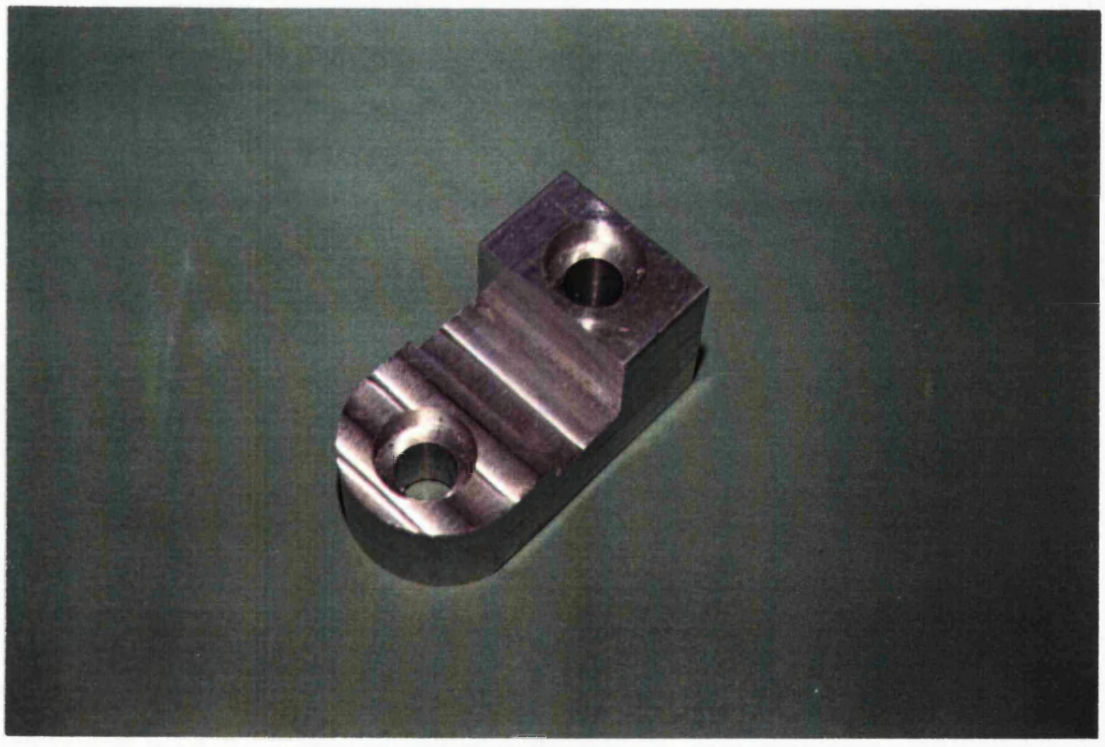

Figure 5.4 The Machined Component

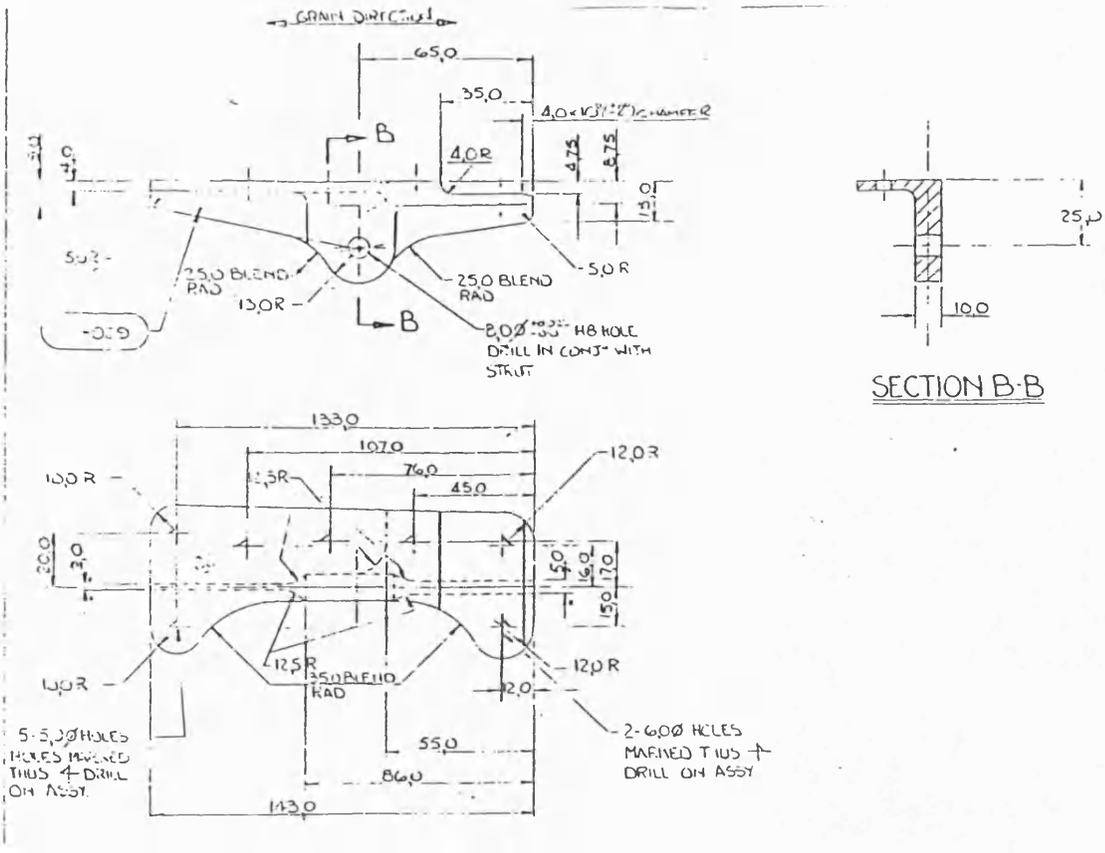

Figure 5.5 A more complicated component

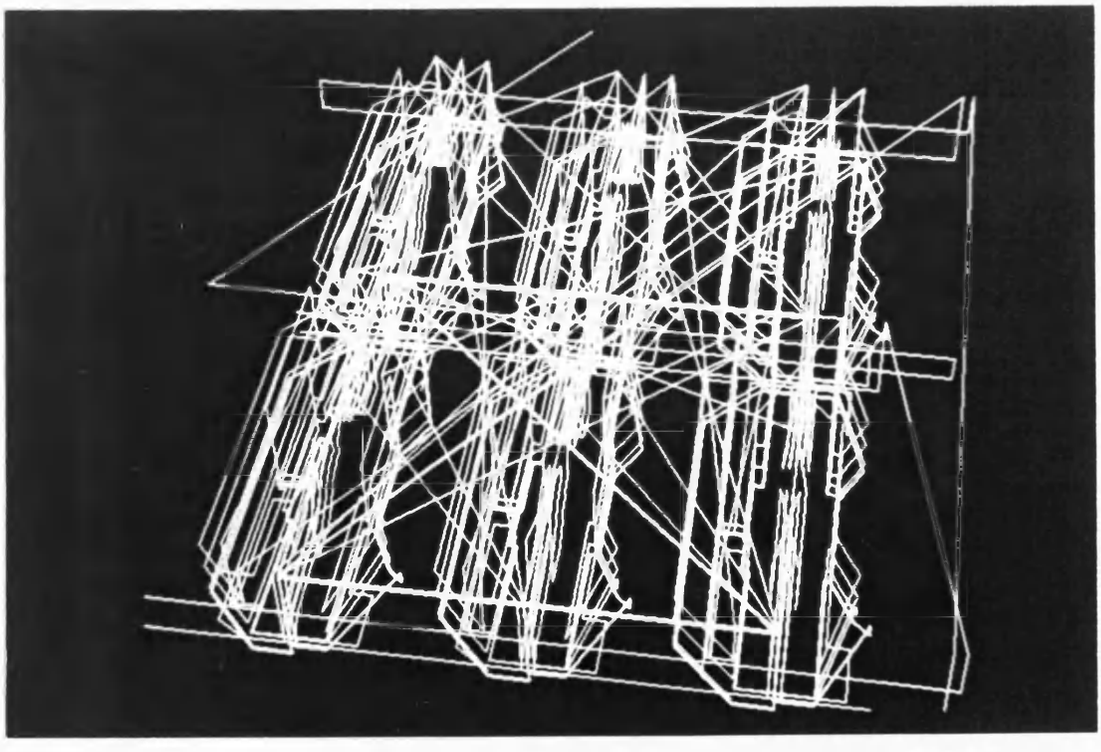

Figure 5.6 Vector-plot of toolpath centre-line

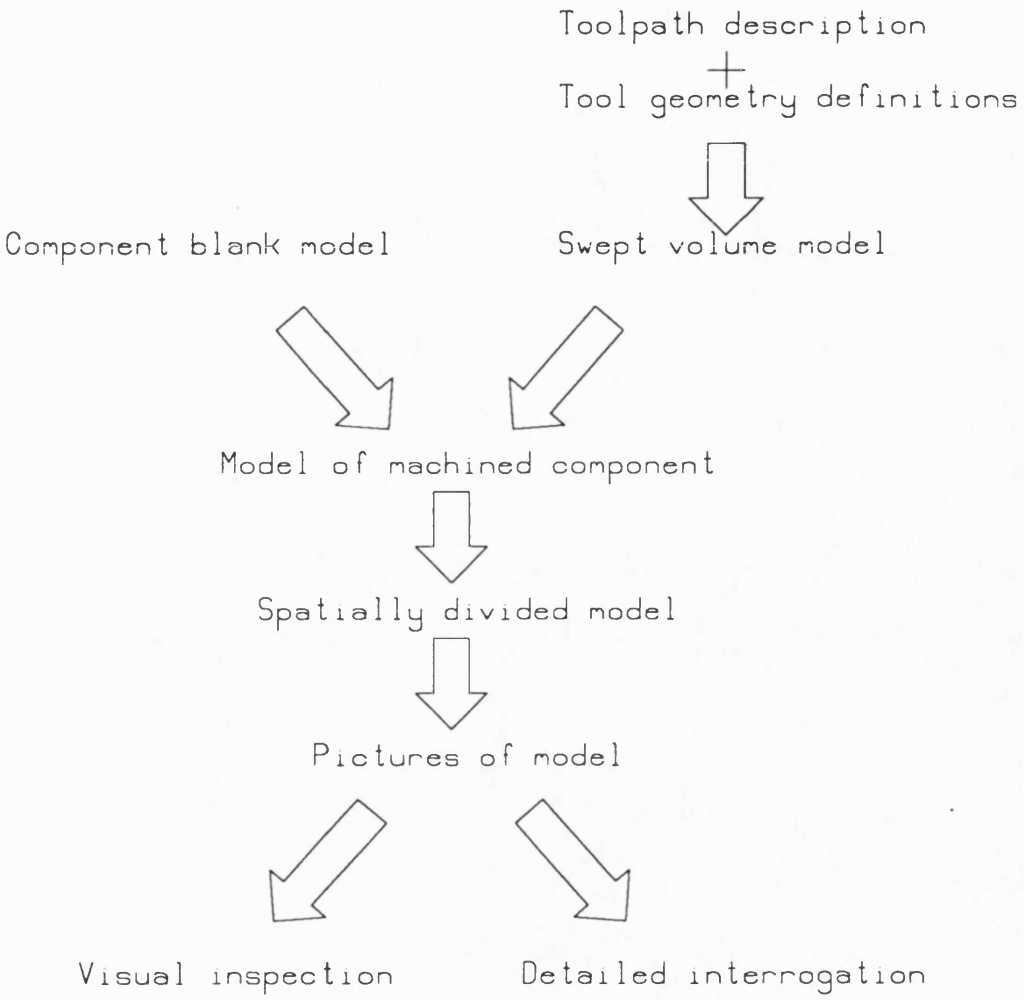

Figure 5.8 The solid modelling approach

CHAPTER 6

Construction of a Solid Model from a Toolpath Description.

The first stage in the toolpath verification system is the construction of a geometric model of the component that results from the machining operation.

The Composition of the Model

This model is composed of a model of the component blank combined with a model of the volume swept by the tool(s) during the machining operation. Other models representing vices, clamps and parts of the machine tool can be incorporated in order to allow the detection of collisions between them and the cutter or component blank.

The model of the component blank may be specified in one of two ways. If a simple rectangular blank is specified (as is the case in many milling applications), then the position and size of the blank may be specified, and the system will then construct the required model. In other circumstances a more complicated model may be needed. This may occur in one of two cases.

The first of these is when the blank has been created by some external process. It may, for example, be a cast or formed object, or it may have undergone a machining process that is not to be verified. In these cases, the blank may be described using a language input SID [30] and/or the graphical input system described in Chapter 4. Pictures of the model may be generated using either the

modeller based on planar half-spaces (DORA [30]), or that based on polynomial half-spaces (DODO).

The second case occurs when a sequence of toolpaths is to be verified and the blank has undergone previous machining that is to be (or has already been) verified. In this situation the model of the 'blank' will itself have been created by the software described in this chapter. When the machining process uses multiple set-ups in which the component is repositioned or inverted between machining operations, the model of the blank is suitably rotated and translated before it is included in the new model. These operations are easily performed using the model definition language.

Models of objects that do not move relative to the component during machining (parts of the machine bed, vices, clamps etc) may also be described using language or graphical input, and unioned with the blank at this stage.

The rest of this chapter describes the process of generating the model of the volume that is swept by the tools for the toolpath that is to be checked. The strategy adopted when generating this model is to create a model for the volume swept by each tool movement separately, and then to combine these to form a model of the total swept volume.

Factors Affecting the Geometry of Machined Surfaces

As stated above, each tool motion is modelled separately. Ideally, the verification system would be capable of modelling any tool shape and path. In practice the range of shapes and paths is restricted by the geometric modeller that is used. In

the project described in this thesis two modellers were used (as explained in Chapter 1), a *facetted* modeller and a *polynomial* modeller. For the verification systems based on either modeller the choice of tools and motions that will be processed is based on the surfaces that need to be represented by the system in order to model each motion. The geometry of the surface produced when a cutting tool is moved relative to the workpiece is dependent on two factors.

The first of these is the shape of the cutting tool itself. The tools most commonly used in a vertical milling machine are end-mills, slot-drills, fly-cutters, centre-drills and drills. (If sculptured surfaces are to be cut, then ball-ended slot-drills are also used). If it is assumed that the tools always rotate sufficiently fast relative to the feedrate then, for the purposes of toolpath verification, each shape of a tool may be represented as the shape of the volume swept by the tool as it rotates about its central axis. (It should be noted here that the model for each tool movement is *not* generated by sweeping a model of the tool along the path for the cut [55], but rather is generated directly.) Hence the end-mill and slot drill may be represented as a cylinder, and the drill as a cylinder and a cone. A centre-drill may be represented either in the same manner as a drill, or as a more complicated assembly of cylinders and cones. The fly-cutter may also be represented, slightly less accurately as a cylinder. A ball-nosed cutter may be represented as a cylinder and a hemisphere. Figure 6.1 shows models of a slot-drill, a twist-drill and a ball-noded slot-drill.

The basic shapes of milling cutters are sometimes modified by adding corner radii chamfers. In some instances it may be desirable to incorporate these parts of

the tool shape in the tool model. If this is the case, then the additional surfaces that would be required are the torus and cone respectively.

The motion of the tool relative to the workpiece also affects the shape of the surface produced by any individual cut. Modern vertical milling machines are capable of one-axis, two-axis or three-axis linear movements and two-axis circular movements (as well as more complicated three-axis movements such as helical interpolation and also four or five axis motions). The surface forms generated when the different surfaces used to model the cutter types are moved along a range of paths are shown in figure 6.2. Note that the cylinder, cone and torus primitive are assumed to have their axis of symmetry oriented vertically. (The shape of the original blank only affects the edges of the cut surfaces. This is of no importance in either of the set-theoretic modelling schemes since they do not store edge information.)

For the facetted modelling system, it was decided to limit the range of tool types and motions that would be processed to those which would generate only singly-curved surfaces. This is because it was considered that too great a number of facets would be required to model doubly curved surfaces to the required degree of accuracy. The system was implemented to handle tool motions corresponding to two-and-a-half axis machining, and will therefore process horizontal linear motions, vertical linear motions and horizontal circular motions for cylindrical tools; and also vertical linear motions for cylindrical and conical tools.

In the case of the polynomial system the limitation is one of the degree of the polynomials generated by a given tool/motion combination. For reasons of

numerical stability, the system will handle polynomials of degree less than 14. Referring to figure 6.2, it may be seen that the scheme is capable of modelling all surfaces that may be generated by the toolpath generation system. In practice, restrictions were made on the range of motions that were processed due to the available time. Thus the polynomial system will process both linear motions in any direction and horizontal circular arcs motions for any of the tool types shown in figure 6.1.

It should be noted that for both systems, any restriction in tool movements applies only to each separate tool movement. There is no reason why the workpiece (and model) should not be translated or rotated between cuts. The full range of valid tool shape and motions is summarised in figure 6.3.

The Toolpath Description

Toolpaths that are to be verified may be described in one of two ways; either as a CLdata file [56](the format of which is defined by the BS 3625 and ISO/DIN 3592 standards), or as a 'machine-level' part-program file that has been generated for a particular machine-tool and/or controller.

The CLdata file is a standard description of the path followed by the tool centre (a point on the axis of rotation of the tool, usually positioned at the level of the tool base), together with non-geometric information such as tool changing, speed and feed rates etc. The file is composed of a number of data blocks. Each block has a record type that defines the class of instruction that the block contains and a sub-type that defines the exact function. These are followed by a number of

parameters, the meaning of which are dependent on the record type. In the case of a linear cut, for example, there are a variable number of parameters, in triples, each of which contains the *(x,y,z)* coordinates of a point through which the tool centre will pass (ie the single record can define more than a tool motion).

CLdata files are generated by many computer assisted part-programming systems, GNC [48] and APT and its derivatives [57] for example. Such files contain a non machine-specific description of the toolpath which is normally *post-processed* to generate a set of instructions for a specific machine-tool and controller. This is essentially a translation process, which takes into account both the required language for the machine tool controller, the geometric layout of the machine axes, and any limitation of the machine as regards tool motions. (The post-processor for a machine not capable of circular motion would, for example, approximate any circular tool movements into a number of straight motions.) Hence if the CLdata file is used as the toolpath definition, the toolpath verification system input is independent of any particular machine, although it should be noted that it will *not* detect any errors that occur during post-processing.

The model-building stage of the toolpath verification system processes the CLdata file sequentially, one block at a time. After reading a block from the CLdata file its first action is to decode the record type. Records may be classified into four groups for the purposes of the verification system. These are:

- Cutter selection.

- Cutter movements suitable for processing by the verification system.
- Cutter movements *not* suitable for processing by the verification system.
- Other instruction, such as the selection of spindle speeds and feedrates.

If the record is a tool selection command, then the new tool diameter and length are extracted, and the tool number and CLdata block number recorded. The CLdata file does not contain the type of tool. This information may be obtained from some other source, or possibly from an 'operator comment' record that accompanies the tool load record. (The information must be available as it will be required by the machine tool operator)

All other non-movement commands are ignored by the model generator, except for coordinate offset instructions.

If the record is a tool movement instruction then the record sub-type is checked to see if the motion is a linear or circular arc. The end point of the movement is extracted and, in the case of the circular arc, the position and orientation of the circle centre axis and the direction of motion (clockwise or anticlockwise). The data are now checked to see if they are suitable for processing by the system. If not(for example they represent a circular arc with a non-vertical axis) then an error is flagged. If the tool motion is capable of being modelled then the next stage is to create the model of the volume defined when the tool-shape is swept along the centre-line.

If the toolpath is defined as a machine-level part-program file, then a similar process to that described above for the CLdata files is required. In this case the verification system must emulate the particular machine-tool and controller for which the program is written. Machine level part programs consist of a sequence of blocks, each block containing a number of commands. There is no standard that defines the meaning of *all* the commands used in part-programs; different machine-tool controllers handle instructions in differing ways and the same machine instruction can have a different meaning on two different controllers. (For example, the *I*, *J* and *K* fields for circular motions may be contain either relative or absolute values.) The verification system will handle input for two different controllers, a FANUC 6M controller used at the University of Bath, and also a machining-centre at use at British Aerospace in Preston. The part-program file is processed in a manner similar to the CLdata file. Each block is read in sequentially, the contents for the block are decoded, and tool movement instructions are extracted for model generation. A record is kept of all 'modal' command data as well as information such as the current tool, whether motion is absolute (G90) or incremental (G91), the plane for circles (G17, G18, G19) etc. The full list of commands capable of being processed is given in figure 6.4.

For a machine-level part-program, a separate file is required that defines the size and shape of each tool to be used. Whenever a tool load record is read from the part-program file, the details of the tool type (end-mill, slot drill, drill, ball-ended cutter, corner chamfer or radiused corner), and diameter, height and any length offset are read from the file. If they are not present, the user is prompted to

supply the information.

Generation the Model for the Swept Volume

The simplest way to model the individual tool motions, bearing in mind that they are to be made up from planar or polynomial half-spaces would be first to construct a number of bounded primitives and use these to construct the model of the swept volumes. A cylinder could be constructed with the same diameter as the current tool, together with a rectangular cross-section bar with a width equal to the tool diameter. These could then be combined to create the model for any linear tool motion of the cylindrical tool in the xy plane, or parallel to the z axis. The disadvantage with this approach is that it introduces redundant half-spaces, especially when generating a facetted model. This adds to the complexity of the model and so will increase the time required to generate pictures of it.

Additional redundant half-spaces will be introduced if the end components of the model for each tool motion are the shape of the swept volume of the tool. These half-spaces are always redundant since at the start of each tool motion, the tool *must* be at the same position as at the end of the previous tool motion. Hence that part of the model corresponding to the start of any non-vertical motion may be a single planar half-space aligned with the centre-line of the tool at that position and oriented such that its surface normal lies along the tool motion vector. This is shown in figure 6.5. Obviously the first cut after a tool load is a special case (although this should be clear of the cutting region). The only disadvantage of this approach is that information regarding that region of material that is swept by both

the end of one tool motion, and the beginning of the next is only attributed to the first motion. In practice this does not cause any problems.

The generation of a model corresponding to the movement of a cylindrical tool, such as an end mill or slot drill for each type of motion is now described. The approach adopted when generating the models is never to introduce redundant half-spaces into the model for any tool motion.

Each cut is in fact modelled as a hole in an infinite solid block rather than as a solid volume. The individual models are joined together with the intersection operator, and then the complete model is *intersected* with the model of the component blank. The resulting model has all the half-spaces oriented such that their surface normals point from solid into air (ie it contains no *difference* operators). This fact may be usefully employed during the later processing of the model.

Vertical Tool Movement

The simplest form of tool movement to model is the linear z-axis motion. This may be modelled as a semi-infinite cylinder with its axis along the center-line of the tool position and with a radius equal to that of the current cutter. In the faceted modelling system, this cylinder is constructed by unioning together a number of vertical half-spaces. In the polynomial system it is generated from two vertical half-spaces, in a manner similar to that shown in figure 2.3, except that in this case the cylindrical half-space has solid to its outside. The lower end of the cylinder is a single plane orthogonal to the tool axis and at a z height corresponding to the base of the cutter at the lowest end of the motion (which may be the start or the

end of the motion depending upon whether the tool is being raised or lowered). If a simple tool modelling scheme is used (see also the section on more complicated tool models) then the upper end of the tool need not be modelled. Figure 6.6 shows the half-spaces used to model such a motion together with the set-theoretic model definition.

Horizontal Linear Tool Movement

The next simplest tool movement to model is linear x-y motion. The model for such a tool motion is comprised of two sides, a base, a 'start' end and a 'final' end. The sides are both vertical planar half-spaces, each positioned at a distance equal to the current tool radius from the centre-line of the tool motion, and oriented such that they are both parallel to the tool motion and are positioned one to either side. The base is a horizontal planar half-space positioned such that it is at the height of base of the tool. The start end of the cut is another planar half-space, oriented such that it is perpendicular to the direction of tool movement, and positioned such that it passes through the start point of the tool motion. In the case of the facetted system, the final end of the model for the tool movement is constructed from a facetted semi-cylinder centered around the final position of the tool, with a radius equal to the tool radius. The polynomial system models the final end as a cylindrical. In this case and 'extra' planar half-spaces is required define the model. The model for a linear x-y cut is shown in figure 6.7.

Horizontal Circular Tool Movement

The most complicated cut modelled by the facetted modeller is the x-y circular arc. Depending on the relative radii of the tool and the cut, and also on the angle subtended by the movement, there are five possible shapes of cut that may be generated. These are shown in figure 6.8. If the radius of the cutter is greater than that of the tool center-line and the arc subtended is $\geq$ 360 degrees then the volume swept is a simple cylinder. If the angle subtended is less than 360 degrees then the shape will be a as shown in figure 6.8b. If the cutter radius is less than the radius of the arc followed by the tool then the plan of the shape will be either a hoop or a cylindrically curved bar with a planar face at the start of the tool motion and a semi-cylinder at its end. These ends may intersect or partially intersect with each other depending on the angle subtended by the arc. These shapes are shown in figure 6.8c,d and e.

The model generator detects these different cases and models them differently. In cases (a) and (e), the models may be built simply from two vertical cylinders and a horizontal planar base. The approach used to build models for the other cases varies between the facetted and polynomial modelling systems. For the facetted system, any simple approach will include redundant half-spaces. The technique used to generate the model for such cuts is based on the algorithm used to generate solid models from sketched input described in Chapter 4. It has the advantage that it may be used on cuts that correspond to figure 6.8b, c and d, and in each case, no redundant half-spaces are generated.

The technique is as follows. The outline of the cut consists of three separate parts. The main body of the cut is a flat-ended segment of a circular annulus. The

width of the annulus is the tool diameter and the radius of its center-line is that of the cut. The ends of the cut are a plane and a semi-circle having the same radius as the cutting tool. A set of points are generated around the boundary of the cut. The recursive convex hull algorithm, described in Chapter 4, is applied to this set of points to generate the model of the sides of the cut.

For the polynomial system, cuts of these types are modelled as shown in figure 6.8. Note that the operator used to join the two planar ends of the cut will vary depending on the angle subtended by the tool motion; for angles of less than 180 degrees the ends are *unioned* together, otherwise they are *intersected.* If the radius of the cutter is less than the radius of the tool motion (case (b)), then the inner cylinder may be omitted from the model.

Three Axis Linear Tool Movement

For all of the models describe above, which correspond to 2½ axis cutting, the base of the model is a simple plane. In the case of three dimensional movement this is not the case. The model for a three axis linear motion of a cylindrical tool is composed from eight different half-spaces (see figure 6.9). The two sides and the ends of the model are as for an x-y linear tool movement. The base of the cut has three parts. At the lower end of the tool movement, which may occur at either the start of the end of the cut, the base is a horizontal plane. Elsewhere the base is of an elliptical form. The elliptical half-space is generated in a manner similar to the cylindrical half-space of figure 2.3. It may be defined as follows:

$$base\_e := radius^2 - (\frac{1}{sin(theta)} \cdot base\_2^2 + vhs^2)$$

where

base_e is the elliptical half-space,

radius is the tool radius,

θ is the angle of the tool motion to the horizontal,

base_2 is the angled planar base to the cut,

vhs is a vertical half-space containing the tool centre-line motion.

Modelling Complicated Tools

The preceding descriptions of models for each type of tool motion have dealt with model generation for basic cylindrical tools. Models for cutting tools with more complicated shapes are constructed in a similar manner although they have more complicated base and end elements. The model generator is written so that routines for handling tools of new shapes may be added in an easy and consistent manner. As long as the additions to the tool shapes are constructed from the primitives shown in figure 6.2, then the models generated by tool motions will be capable of being modelled by the 'polynomial' modeller and hence all are suitable for the toolpath verification system described in this thesis.

The tool models described above only model the cutting surfaces of each tool. More complicated tool models may be used if collisions between non-cutting parts of a tool and the workpiece, clamps etc are to be detected. In this case, the additional volume swept by the non-cutting part of the tool is also modelled for each

tool movement. These extra models may be combined with the model for the volume swept by the cutter, but coloured differently to allow them to be easily distinguished later. (Clearly any intersection between such a model and the component model is an error.) In a similar may, additional models of the volume swept by the machine-tool head and spindle assembly may also be added to the tool model.

The additional models needed to represent the volume swept by the tool holders should add little to the complexity of the model after spatial division, since in most cases, the extra model elements will lie within *air* and will be pruned out during division. Those parts of the model that represent the shank of the tool are expected to be pruned-out less well. Statistics of the effect of using more complicated tool models are given at the end of Chapter 7. Figure 6.10 shows a range of more complicated tool models using cylinders, cones and spheres.

Other Features of the Model Generator

As an alternative to generating models of the complete component, models of part of the component may be generated either by initially modelling only part of the component blank, or by intersecting the final model with a required section-plane. Sections may be defined by simple planes or by more complicated surfaces. These allow internal features of the model to be easily seen for verification purposes.

When the complete input file has been processed, the model definition is written out to a disc file. This file contains two parts. The first a list of planar half-spaces. Each half-space is defined by the three direction cosines of the vector that

is the outward-pointing normal to the half-space plane, and the distance from the origin to the plane. Each half-space is also tagged with a colour which may be used later to detect erroneous surfaces created by non-cutting parts tooling, or by rapid tool movements or movements with the spindle not rotating. All half-spaces except those that model the component blank are also tagged with a record of the CLdata file, or part-program file, block-number that contains the tool motion that the half-space partially models. This block number may be used later in the interrogation process to relate surfaces in the model back to the toolpath element that generated them. Following the half-space list is a list of half-space references and set-theoretic operators in Reverse Polish order that describes the model. In the case of the polynomial modelling scheme, numeric constants and *arithmetic* operators are also present, enabling the curved half-spaces to be generated from planar half-spaces.

The model generator also writes out a file that contains a list of the current block number at each tool-load operation. This allows the model interrogator described in Chapter 8 to generate tool information from block numbers.

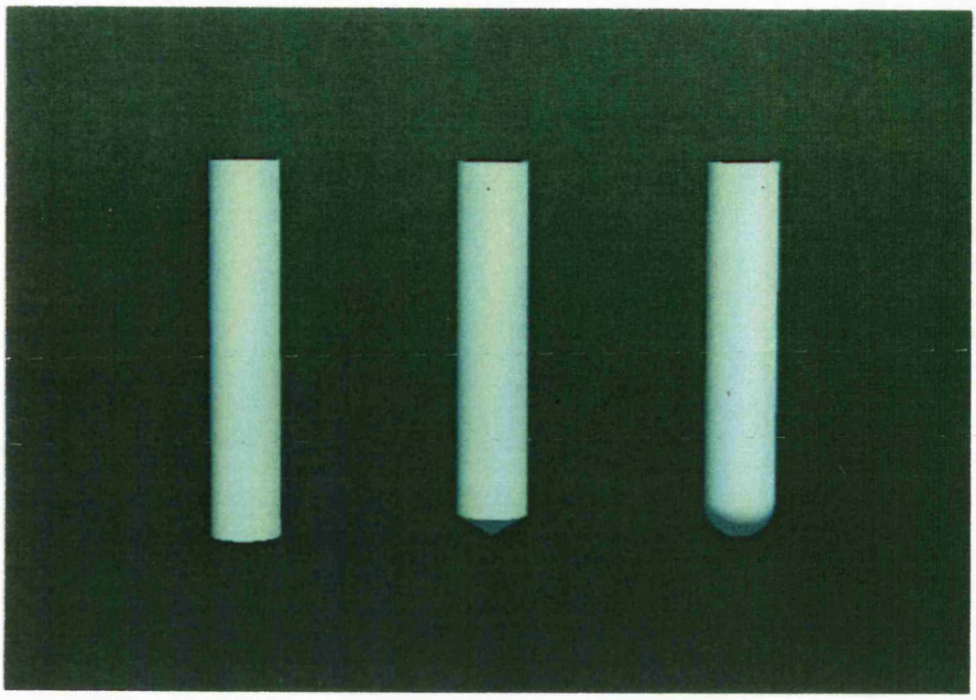

Figure 6.1 Simple tool models

Primitive	Motion type			
	x-y straight	z straight	x-y-z straight	x-y circle
cylinder	plane cylinder	cylinder	plane cylinder ellipse	cylinder
cone	plane cone	cylinder cone	plane cone	cone
sphere	cylinder sphere	cylinder sphere	cylinder sphere	sphere torus
torus	plane cylinder torus	cylinder torus	plane cylinder torus	plane torus

Figure 6.2 Surfaces created by sweeping primitive shapes

Tool	Motion type			
	x-y straight	z straight	x-y-z straight	x-y circle
Twist drill	**F P ***	**F P**	**P ***	**P ***
Slot drill	**F P**	**F P**	**P**	**F P**
End mill	**F P**	**F P ***	**P**	**F P**
Fly cutter	**F P**	**F P ***	**P**	**F P**
Ball-nosed cutter	**P**	**P**	**P**	**P**

Key

F: valid for facetted system; **P**: valid for polynomial system;

*****: incorrect tool motion if cutting

Figure 6.3 Tool motions and shapes handled by the system

G00 - Rapid motion.
G01 - Feedrate motion.
G02 - Circular motion (clockwise).
G03 - Circular motion (anti-clockwise).
G17 - Select xy-plane for circles.
G20 - Input in inch.
G21 - Input in mm.
G28 - Return to reference point.
G40 - Cutter compensation cancel.
G80 - Cancel cycle.
G81 - Cycle drill.
G89 - Cycle inhibit for current block.
G90 - Absolute programming.
G91 - Incremental programming.
G92 - Set origin.
M00 - Program Stop.
M01 - Optional Stop.
M02 - End of program.
M03 - Spindle on (clockwise).
M04 - Spindle on (anti-clockwise).
M05 - Spindle off.
M06 - Tool Load.
M08 - Coolant On.
M09 - Coolant Off.
M30 - End of program.

Figure 6.4 List of G-codes handled by the system

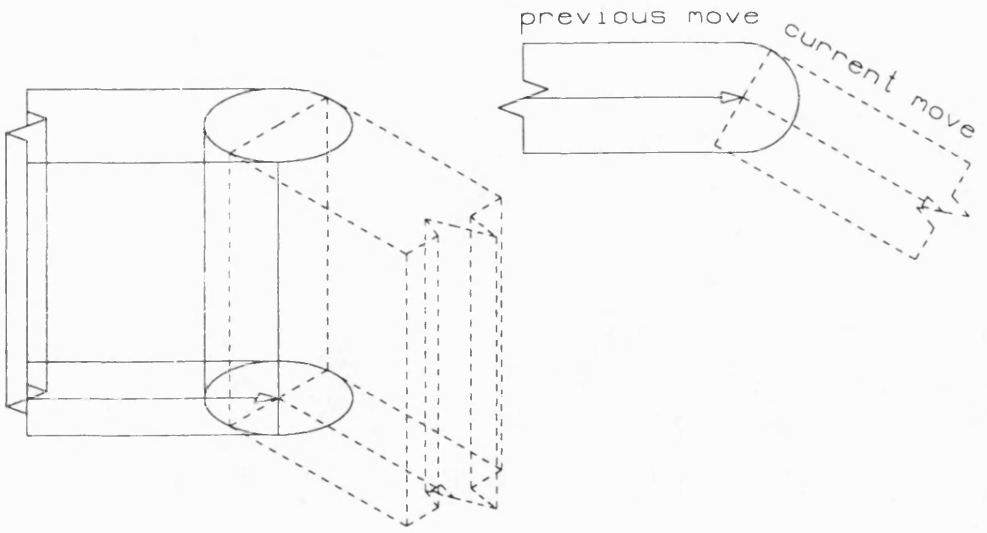

Figure 6.5 Modelling the start of tool motions

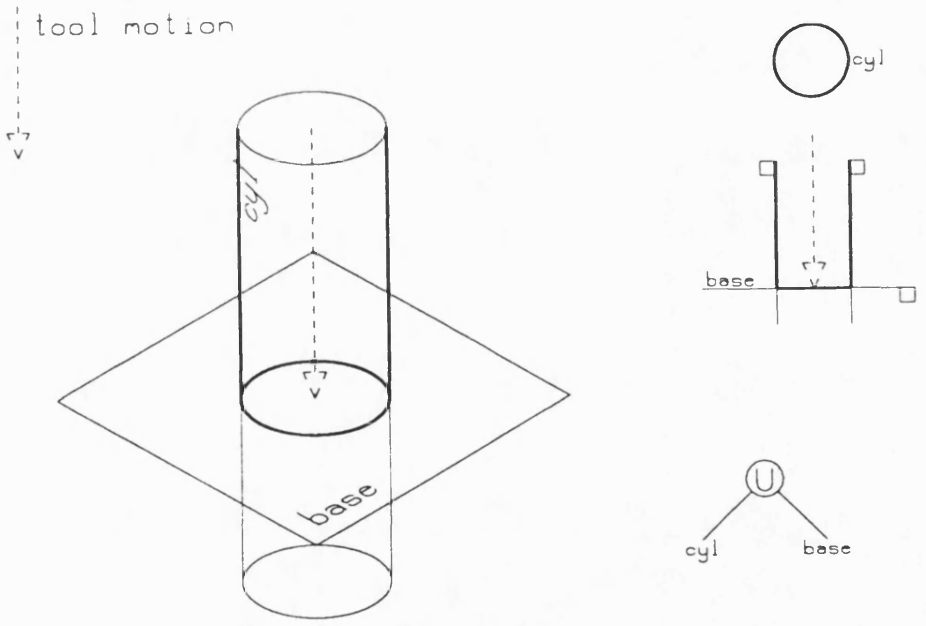

Figure 6.6 Model of vertical tool move

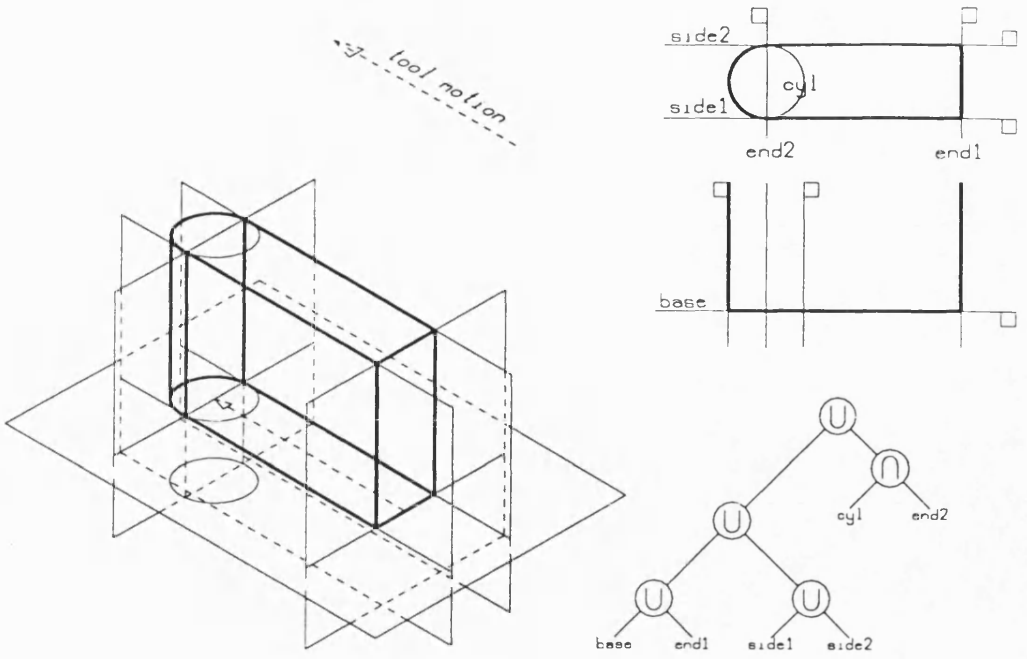

Figure 6.7 Model for horizontal linear move

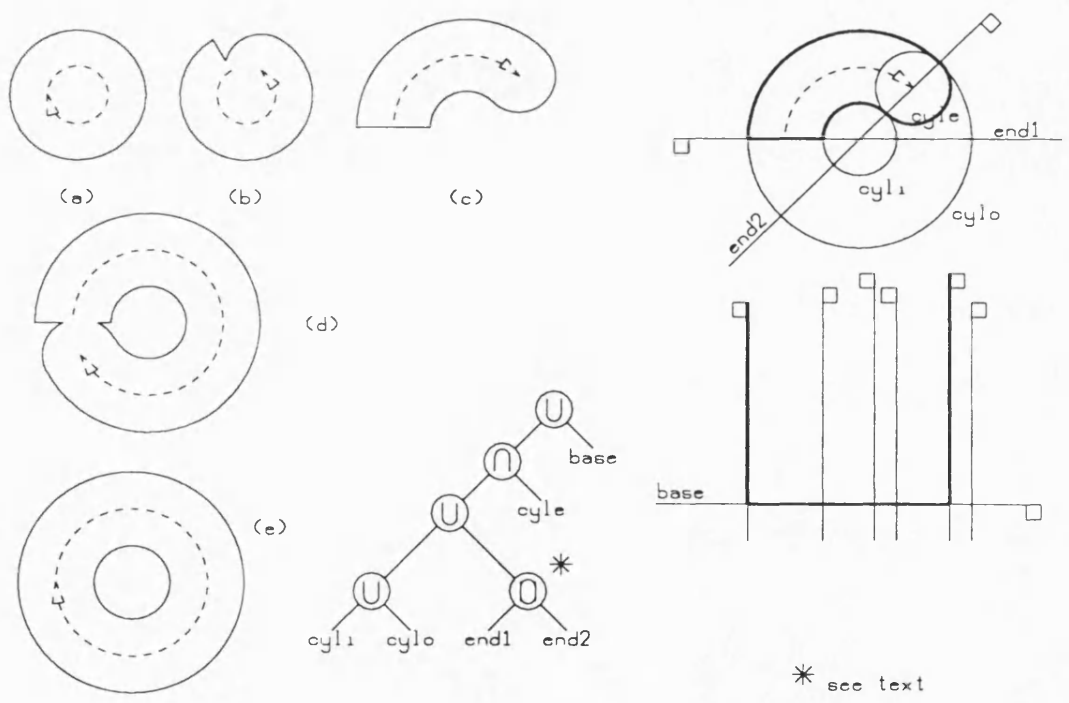

Figure 6.8 Model for horizontal circular move

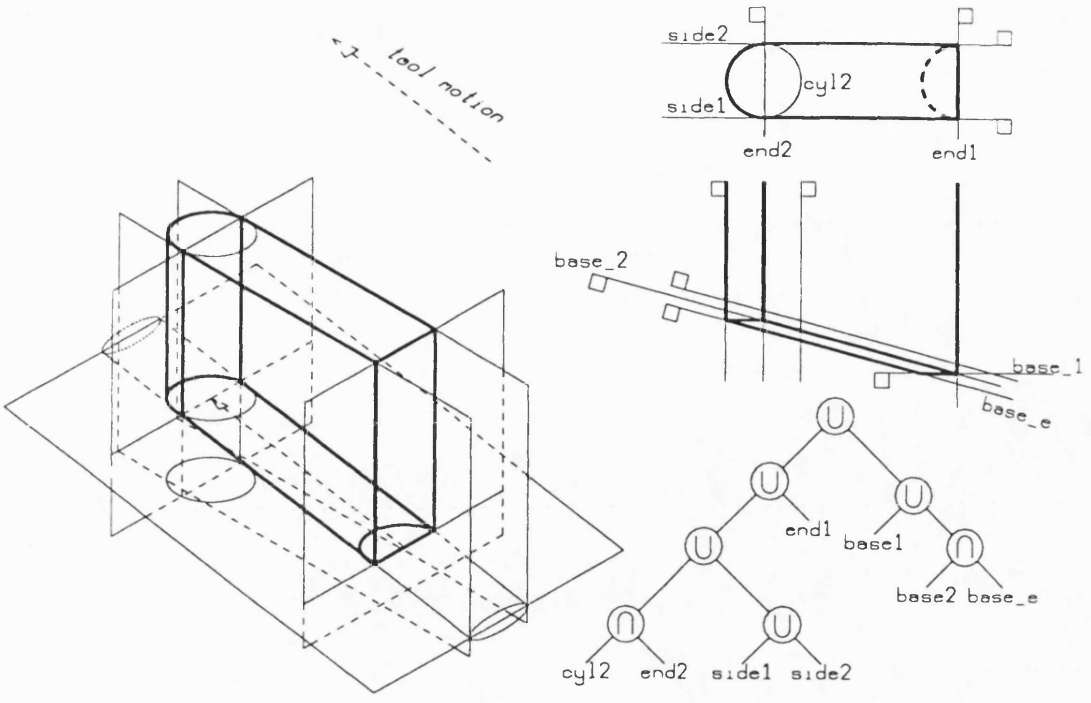

Figure 6.9 Model for three-axis linear move

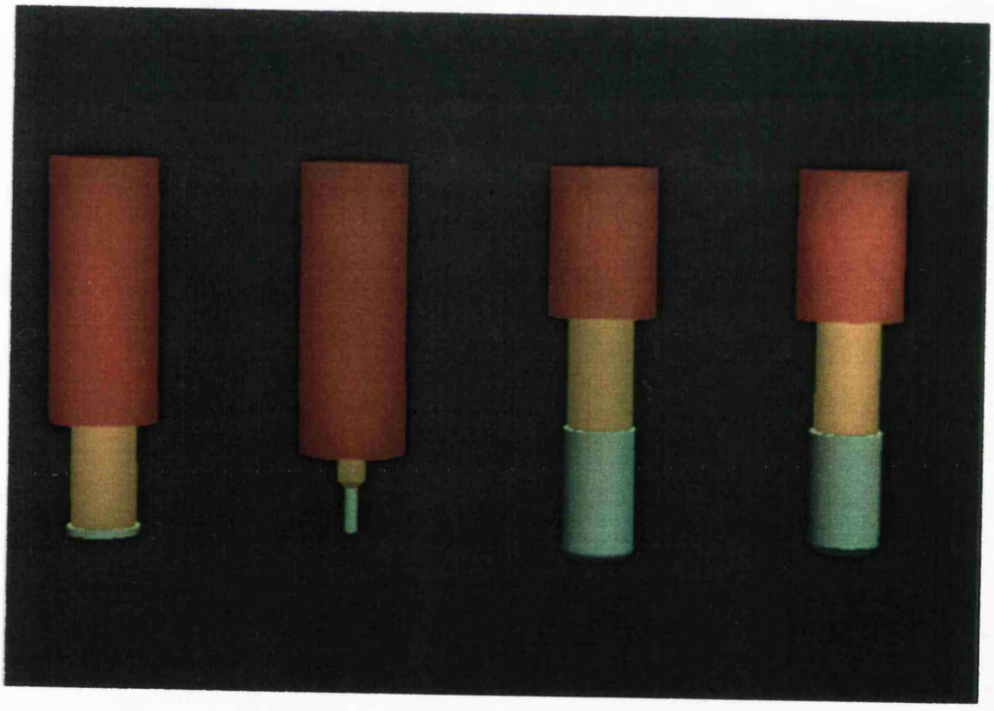

Figure 6.10 More complicated tool models

CHAPTER 7

Creating the Divided Model

The second stage of the verification system is the generation of the spatially divided model from the component model.

Requirements of the Divided Model

As outlined in Chapter 3 there are a number of ways of applying a spatial division strategy; the strategy best suited for any application is dependent on the intended use of the divided model. In the case of the VOLE modelling system for example the division process could be oriented around the generation of a single view. In the toolpath verification system the two uses of the divided model are the generation of several views of the model, and the interrogation of the model using these views. At the division stage the number of views and their orientation may be unknown. However there will almost always be several differing view-points. Hence the division cannot be optimised for a single set of viewing parameters, but should result in a divided model capable of efficient use from any direction.

The model divider generates a binary divided structure with the split-plane dividing the sub-space for each non-leaf node oriented such that the normal to the plane is parallel to either the x, y or z axis.

Both of the uses of the divided structure (image generation and interrogation) use ray casting techniques. This involves projecting a ray from a given viewpoint

into the divided structure. The ray will pass through one or (usually) more sub-spaces. For each sub-space that the ray passes through before it reaches the surface of the model there is an overhead in computation time. Hence it is required to keep the overall number of sub-spaces to a minimum. A potentially more costly effect (from a computational viewpoint) is that of underdividing, since this will result in sub-models that are over-complicated. This is undesirable in any spatially divided model, but it is especially the case in the toolpath models since they contain a large number of half-spaces.

Method of Generation

A simple method of creating a binary divided structure was outlined in Chapter 3. It was noted that it is not easy to control model division so as to create an optimally divided model; two of the problems encountered in creating an efficiently divided model are, first, that it is not possible to tell if the division of a leaf node will lead to a reduction in model complexity until after it has been split; and second, that several levels of apparently unnecessary division may then lead to an overall reduction in model complexity.

In order to overcome these problems it would be useful to be able, at a given stage of model division, to assess the result of further division before incorporating this further division in the divided structure. This could be achieved by creating a tree structure that is capable of being 'cut back' if it is apparent that it is over-divided (the method of detecting such over-division is described later). This is effectively what is done in the model division stage of the toolpath verification sys-

tem. The logical tree-structure used by the divider is shown in figure 7.1. Each node in the tree has a sub-space associated with it and leaf nodes also have sub-models. The nodes in the tree can be categorised into those that are definitely part of the divided model, (*divided-model nodes*), and those that are still being considered for possible further division or cutting back. This second category of nodes will be referred to as *sub-tree nodes* since they form a number of distinct *sub-trees* within the overall model tree. The root node of each of these sub-trees is the equivalent of a node being considered for division in the simple division process described in Chapter 3. That simple process can now be modified so that at each stage in the division process one of the following actions is performed:

1 Transfer the root node of a *sub-tree* into the divided model as a non-leaf node and thereby create two new *sub-trees*, or

2 Write the root node of a sub-tree into the divided model as a leaf node (and discard the rest of that *sub-tree*), or

3 Split one of the leaf nodes of a *sub-tree* to form the two new nodes. (The sub-model for each new leaf node is formed by pruning the sub-model for the old sub-space to each new sub-space.)

Although the data-structure used in the division process is logically a single tree that is grown and cut-back to form the final divided model, it is physically stored as two separate types of tree structures. This is done because of the differing storage requirements for the *divided-model nodes* and the *sub-tree nodes* (more fully described later). Also, the *divided-model tree* may be considered to be

a write-only structure for the division process, the *sub-tree trees* are read-write structures. Note that each sub-tree is completely independent of the others. Therefore they can be processed individually. The logical data structure, shown in figure 7.1, may be modified to reflect this, and now consists of three categories of trees as shown in figure 7.2:

- The partially complete divided model (the *divided-model tree*).
- A number of *sub-trees* awaiting processing (the co-trees).
- The *sub-tree* currently being worked on (the *current sub-tree*).

The Division Process

The method used to generate the spatially divided model is based on the simple binary tree generation method already described in Chapter 3. The division process starts by initialising the *current sub-tree* to contain a single node representing the object-space and model which is obtained from the model generator described in the previous chapter.

The division process proper can now commence. The process repeatedly performs one of the actions listed above, which can be re-stated in terms of the data-structure of figure 7.2. All actions are performed on the *current sub-tree*. They are shown diagrammatically in figure 7.3.

- The *current sub-tree* is 'beheaded' (see figure 7.3a): its root node is transferred into the *divided-model tree* as a non-leaf node thus creating two new *sub-trees*, one of which is added to the the list of co-trees, the other

becomes the *current sub-tree*. (If the *current sub-tree* has only one node then it cannot be beheaded.)

- The current root-node is written into the divided model as a leaf node (see figure 7.3b). One of the co-trees is picked as the *current sub-tree*. If there are no more sub-trees to be processed then division is complete.
- One of the leaf nodes in the *current sub-tree* is split to form two new leaf-nodes (see figure 7.3c).

Controlling Division

A control mechanism is needed to enable the model divider to choose which of the above actions to perform, and in the case of the second action, which node in the *current sub-tree* to split. This mechanism is analogous to the sub-space testing mechanisms discussed in Chapter 3. Since the only process that is to use the divided model, both for picture generation and later interrogation, is raycasting, the control mechanism can be tailored to this process.

Several control strategies have been tried. Each is based on storing additional information for each *sub-tree* node representing various computational loads associated with that node's sub-space and sub-model, the overall goal of the control strategy being to minimise the total evaluation load associated with the *divided-model tree*. A simple strategy, used in the facetted verification system is explained in [58]. The meaning and method of calculating the loads are discussed later. First consider the effect of the three actions described above.

The first action to be considered is beheading. If the computational load for evaluation of the current root node is greater than that at a lower level (ie a more-divided state) in the *current sub-tree* then it is clear that the current sub-space and sub-model would be better incorporated into the divided structure at a more divided state. The action that achieves this is beheading.

If that option is not chosen then the choice of action is either to write out the current root node, or to further divide the *current sub-tree.* This choice is not so clear-cut as the previous one, since the effect of further division of the *current sub-tree* in not known. Indeed, this 'trial' division may be regarded as an information gathering action.

The control mechanism must control the amount of this trial division. If too little is performed then the divided structure may be less efficient than otherwise possible. If too much trial division is performed then the size of the *sub-trees* will become excessively large; and the extra time taken by the division process will possibly outweigh time saved at the raycasting stage.

One possible strategy is to consider the load associated with the *current root-node* compared with the size of the *current sub-tree.* In this way, the maximum level of trial division can be set for any given current root-node and so division that can result only in minor savings can be curtailed. Another is to look at the amount of simplification that has resulted from the current level of division. If little or no simplification has occurred then it may be best to write the current root node as a leaf node in the divided model.

The chosen strategy combines both of these; trial division is allowed to continue until the number of nodes in the *current sub-tree* exceeds a limit which is calculated from the evaluation load of the current root node and the ratio of this load, to the minimum load at a lower level in the *current sub-tree* (or until the *current sub-tree* is beheaded).

Hence division is controlled using two tests. The first test compares the evaluation-time load for the current root node and the minimum such load for the lower levels in the *current sub-tree*. The second test compares the size of the *current sub-tree* with a limit based on the two loads used in the first test. This strategy may be summarised as:

If ($L_{lower} < L_{root}$)

behead current sub-tree

Else

{

If ($nnodes < \alpha \,.\, L_{root} - \left[1 - \beta . \left[\frac{L_{lower}}{L_{root}} - 1\right]\right]$)

split leaf-node in current sub-tree

Else

write root-node of current sub-tree into divided model

}

where

nnodes is the number of nodes in the *current sub-tree*,

L_{root} is the evaluation load for the current root node,

L_{lower} is the minimum evaluation load for the *current sub-tree* at a level

below the root node,

α and β are tuning constants

This is shown graphically in figure 7.4. The graph may be divided into 3 regions, representing the 3 actions. The number of nodes in the *current sub-tree* is plotted along the horizontal axis and the ratio $\frac{L_{lower}}{L_{root}}$ along the vertical axis. The status of the root node of the *current sub-tree* can be represented by a point on the graph. If a *current sub-tree* is created containing a single node, its status may be plotted as a point on the *nnodes* = 0 axis. As the sub-tree is split the number of nodes increases and point moves to the right. If the complexity of the sub-models for nodes at lower levels in the *current sub-tree* decreases, then the height of the point above the $\frac{L_{lower}}{L_{root}} = 1$ axis decreases. If it passes below the $\frac{L_{lower}}{L_{root}} = 1$ axis then it is definitely better to behead the tree. Otherwise it must eventually pass into the 'write-out' region and the current root-node is written into the divided structure. In this way a limit is placed on the size of the current sub-tree. As the ratio of L_{lower} to L_{root} approaches unity, then the maximum size *current sub-tree* increases. Hence if it appears likely that it is going to be best to behead the tree, then exploratory splitting is allowed to continue for longer than if no reduction in complexity is noticed.

In the expression that controls the size of the *current sub-tree* there are two constants. The first of these (α) controls the maximum size of the *current sub-tree* (for any given root node sub-model complexity). As the value of α in increased so

does the size to which the *current sub-model* is allowed to grow. The second constant (β) controls the effect that lack of simplification has on curtailing this size. As its value is increased, then the maximum size of the *current sub-tree* is reduced when little or no simplification results from this 'trial' division. In terms of the graph of figure 7.4, increasing α moves the split/write-out boundary to the right, increasing β swings the boundary in an anticlockwise direction. The effects that changing the value of the constants has on the division process for an actual model are discussed at the end of this chapter.

If the decision is to split a leaf node, then a suitable leaf node in the *current sub-tree* has to be selected. The system chooses the node which has the largest computational load associated with it, since this is the node likely to yield the greatest reduction in complexity after splitting. The node is found by traversing the *current sub-tree*; a process that is not overly inefficient since the tree will usually be small.

Creating the new Sub-spaces

The sub-spaces for new leaf-nodes are always generated by splitting the sub-space for chosen leaf node into two new sub-spaces. Four strategies for determining the location and orientation of the split plane were investigated. The first of these always splits the sub-space into two halves along its longest side.

The second strategy splits the sub-space into three separate pairs of new (temporary) sub-spaces. The split-plane for each pair passes through the centre of the

sub-space of the leaf node. The three split-planes are oriented parallel to each of the *xy*, *xz* and *yz* planes. After pruning, the pair of sub-spaces with the lowest total load value are retained, and the remaining two pairs are discarded.

In the third strategy, the sub-space is split into nine pairs of new sub-spaces. The split-plane is tested at locations ¼, ½ and ¾ of the way along each side of the leaf node sub-space. Again, the pair of sub-spaces with the lowest total load value are chosen. A similar scheme is used in the forth strategy, the split-plane being tested at seven locations along each side of the sub-space, generating twenty-one pairs of sub-spaces.

The Load Function

The only computational load that is required for each *sub-tree* node by the control strategy described previously is the 'evaluation-time' load which the node would contribute to the overall computational load incurred when generating raycast pictures from the divided model if the node became a leaf-node in the divided model.

The evaluation-time load associated with a leaf node in the *divided-model tree* is dependent on two factors: the geometric complexity of the node's sub-model, and the likelihood of a ray passing through the sub-space. The evaluation process for the node, which is described in the next chapter, consists of finding the intersection points of the ray with each half-space in the sub-model, ordering them, and then performing a membership-test on each point in turn until either a real surface is found, or all points have been tested.

The load for finding the ray intersections with a polynomial half-space is dependent on the degree of the polynomial: the higher the degree, the larger the load to find its roots. The time to insert the points in a list is of order $n \log n$ were there are n points. The time taken to perform a membership-test on a point is of order n, and on average, half the points will need to be tested. The number of roots for a polynomial half-space is dependent on the order of the half-space.

Several load-functions were tested of the form:

$$L = \sum_{i=0}^{nhs} (D(P_i))^a \cdot nhs^b$$

where

L is the load for a *sub-tree* node,

$D(P_i)$ is the degree of the polynomial half-space,

nhs is the number of half-spaces.

a and b are constants

In practice the algorithm was found to be relatively insensitive to the exact load function. The function finally chosen defines the load as:

$$L = \sum_{i=0}^{nhs} (D(P_i))^2 \cdot nhs$$

The likelihood of a ray passing through the sub-space is assumed to be proportional to the surface area presented by the sub-space to the ray vector. This is proportional to the total surface area of the sub-space. In practice, the expression used for this factor is:

$$area = x_{side} \cdot y_{side} + y_{side} \cdot z_{side} + z_{side} \cdot x_{side}$$

where x_{side}, y_{side} and z_{side} are the lengths of the sides of the sub-space.

Thus the load value for a node may be expressed as:

$$L = \sum_{i=0}^{nhs} (D(P_i))^2 \cdot nhs \cdot area$$

Maintaining the Load Values

In order to choose which action to perform, the division program has to compare the load value for the current root node with values at lower levels in the *current sub-tree*. This could be done by searching through the *current sub-tree* whenever a decision is to be made, and calculating the minimum total value for any given combination of nodes whose sub-spaces fill the volume of the current root node's sub-space. This would be computationally expensive as there are many possible patterns of division to be considered. This inefficiency can be avoided by keeping (for each non-leaf node) a record of the minimum total load value for the tree below the node.

Note that the values only change if a leaf node in the sub-tree is split. When this happens, the only nodes that will be affected are those that lie on the branch of the sub-tree between that leaf node and the current root node. Therefore all that is needed to maintain the values is to work from the leaf node back to the root, calculating the updated value for each node by adding the load values of its son nodes.

When combining the load values for the two son-nodes of each node, a constant may also be added. This constant represents the computational load incurred during the tree-descent stage of the evaluation process.

The Data Structures used in the Division Process

The data-storage requirements for nodes in the *divided-model tree* and those in each *sub-tree* differ. Each *sub-tree* node has to store the limits of its sub-space. The sub-model valid for that sub-space also has to be stored since any *sub-tree* node may become a leaf node in the *divided-model tree*. The nodes also need to contain pointers to maintain the tree structure.

Each sub-model consists of planar half-spaces (and arithmetic operators in the case of the polynomial modelling system) and set-theoretic operators. The sub-models are stored as a Reverse-Polish ordered list of operators and references to the planar half-spaces. Each half-space requires several fields to define its geometry and also other information relating to its generation. Each half-space is likely to appear in more than one sub-space, and, as the model is pruned to each new sub-space, the half-spaces are not modified. Hence storing the model as references to a list of half-spaces is more efficient than incorporating the half-spaces themselves in the model both in terms of data storage and computational efficiency.

The sub-models for leaf nodes in the divided model are not altered. Those for *sub-tree* nodes need to be copied and pruned if the node is split, or deleted if the *sub-tree* root node is written into the divided model. Extra 'temporary' pairs of sub-models are created by the leaf-node division strategies that create multiple

pairs of sub-spaces. During the division process a large number of such sub-models will be generated and deleted. Hence some method of storing them dynamically is required. The size of the models varies and the storage method should allow for this. The technique used is to store the sub-models for all of the sub-tree nodes in a single list, the RP-list. Each sub-tree node has a pointer to the start of its sub-model record in the list. The sub-model record contains its length (which may be used to find the start of the next record) and a pointer back to the node, followed by the sub-model for the node.

Each sub-space entry in the sub-trees requires the same amount of storage. They are therefore most conveniently (and efficiently) stored in a number of one-dimensional arrays. As the sub-trees are deleted the storage for the nodes in it must be released. A free-list is used to access the unused entries.

As stated previously, each *sub-tree* is processed sequentially. A list of pointers to their root nodes are stored in a stack. Each entry in the stack also has a pointer to the node in the *divided-model tree* where the *sub-tree* is to be located when it is written out.

The only data-structure in the model divider that requires dynamic memory allocation and deallocation is the RP-list. (All other data-structures use either free-lists or stacks.) As *sub-tree* nodes are deleted, their sub-models are removed from the list. This results in gaps. Two possible ways of reusing these gaps are either to maintain them in a data-structure that reissues them as required; or to always add new sub-models to the end of the RP-list, and then to 'garbage-collect' the list when it becomes full.

The first solution has two disadvantages. Firstly, adjacent sections have to be coalesced, which may be computationally expensive. This is especially so since there may be a large number of gaps. Secondly, the amount of storage required for a new sub-model is not known in advance. The second solution was thus used. Since there are likely to be a large number of gaps in the list, compared to a few active sections, a copying garbage-collection scheme is used.

The Structure of the Divided Model

A diagram of the tree-structure of the *divided-model tree* is shown in figure 7.5. Rather than store the position and size of each sub-space in the tree, each non-leaf node in the tree contains a record of the the position and orientation of the *split plane* that divides its two sons. This is more efficient both in terms of memory-usage and also (as will be seen in Chapter 8) for raycasting into the model. Nodes are written into the divided-model such that the sub-space for the right son of each divided-model non-leaf node is 'more positive' than that for the left son. This increases the efficiency of the tree-descent code used when ray-casting.

The *object-space* for the model is also stored in terms of its x, y and z limits, as is a list of half-spaces that form the model. Leaf nodes contain that part of the model that is valid for the sub-space as a reverse-polish ordered list of half-space references and set-theoretic (and, in the case of the polynomial system, arithmetic) operators.

Half-space Pruning

The exact pruning method differs between the two modelling schemes. The method used in the planar system utilises the fact that, for any sub-space and sub-model, each half-space in the sub-model must pass through the sub-space. In order to categorise the contribution of any half-space to each of the two son-nodes of a node, the half-space may be compared with the plane that splits the two sons. The intersection of this split plane and the sub-space forms a rectangle. The signed distance from each of the four corners of the rectangle to the half-space is calculated. If all the distances are positive, then the half-space contributes *air* to one of the new sub-spaces, and passes through the other. Similarly, if they are all negative, then the half-space contributes *solid* to one of the new sub-spaces, and passes through the other. If some of the distances are positive, and some are negative, then the half-spaces passes through both new sub-spaces. Having so classified each half-space, the two new sub-models may be generated by pruning the sub-model using a scheme similar to that described in Chapter 2.

The pruning system for the polynomial half-space system is similar, although the method used to categorise the contribution of each half-space is different. The classification technique, which is uses *interval arithmetic* is described in detail in [14]. The classification uses the polynomial half-space definitions which are in terms of planar half-spaces and arithmetic operators. For each such planar half-space, the interval of distances from the half-space to the limits of the new sub-space is found. These distance intervals are combined using the operators in the arithmetic half-space definition and interval arithmetic [59] . This results in a

distance interval which is valid for the polynomial half-space and the sub-space. If this interval spans zero then the half-space may pass through the sub-space; if the interval is entirely positive the half-space contributes *air* to the sub-space, otherwise it contributes *solid.*

The Performance of the Model Divider

The divider has be tested on a range of toolpath models varying in size up to a maximum of 31597 half-spaces for the second component example given in Appendix 1.

Table 7.1 shows the effect that varying the value of the tuning parameter α has on the division of the toolpath model for the first example component shown in Appendix 1. The model contains 564 half-spaces, of which 443 are planar, the remaining being polynomial surfaces of degree 2.

As the value of α is increased so does the maximum size (and average size) to which the *current sub-tree* grows before it is either beheaded, or its root written into the divided model. Allowing the size *current sub-tree* to increase results in 'better' division of the model. Hence the total load (the sum of the loads calculated for all the leaf nodes in the divided model) decreases. The better-divided models are bigger, as measured both by the number of nodes in their tree-structure, and also the amount of memory required to store them. The time taken for the division process increases with the amount of 'trial' division performed.

Table 7.2 gives the equivalent statistics for the model dividers that split leaf-nodes in the *current sub-tree* into more than one pair of sub-spaces and then

choose the pair that give the minimum load. For the same value of α, the division times are larger than those in table 7.1. This is because whenever a node in the *current sub-tree* is split, the sub-model pruning must be performed several times. The times are not, however, three, nine and twenty-one times larger than those for the first divider since, in general, the average *current sub-tree* sizes are smaller than those for the first divider (for any given value of α). Figure 7.7 shows the division times for the four schemes plotted as a function of α.

The reduction in the average size of the *current sub-tree* before the root node is written into the divided model can only be caused by a reduction in the average load for such nodes. The reduction in the average size of the *current sub-tree* before it is beheaded suggests that less 'trial' division of the *current sub-tree* is needed before the minimum load at some lower level falls below that for the root node.

Comparing the 'Total Load' values for the four division schemes, it is apparent that as the number of choices of sub-space split-planes is increased, the total load for the divided-model decreases (for the same value of α). This is consistent with the reduction in the average load for nodes in the *current sub-tree* suggested above.

Figure 7.6 shows a graph of division time plotted against total load in the divided model (from the data in tables 7.1 and 7.2). It may be observed that the multi-way dividers are capable of generating divided models with a lower total load. For models that are to be divided to a lesser extent, the simple divider may generate a divided model with a lower load.

It would thus appear that for applications where much use is to be made of the divided model, the extra time taken by the multi-way dividers may be recovered if it results in a reduced computational time for the application. This should occur if the load calculation is a valid measure of the computational load incurred by the application. The validity of the load function is discussed at the end of Chapter 8.

It may be noted that at the initial stages of division, the direction of split is often immaterial, and it is at this stage that the cost of the 'best of n' strategies are greatest. It may therefore be advantageous to use a regular division strategy for the first few levels of division, and use the more complicated strategies at lower levels.

Table 7.3 shows the effect that using more complicated tool models has on the division process. The same toolpath as generated the model whose division statistics are given in tables 7.1 and 7.2 was used with more complicated tool models (each tool modelled as three concentric cylinders). This resulted in a model of the swept volume containing 1687 half-spaces, of which 1324 are planar. For the corresponding values of α, the divided models based on the complicated tool models have higher loads, and are larger than those based on the simple tool models. As the level of division (controlled by α) is increased, the additional load caused by the complicated tool models is reduced. This may be attributed to the fact half-spaces that lie in 'air' are more likely to be pruned out as the level of division is increased. If collisions between non-cutting parts of the tool and the component do not occur, then all of the additional half-spaces in the complicated

tool model will lie in air.

Table 7.4 shows the effect of changing the parameter β for the same model used to generate the data in table 7.1. Altering the value of β would appear to have little advantage over varying α, in terms of the affect that it has on both the division process or the divided model.

α	Max. (Write)	Ave.	Max. (Behead)	Ave.	Time (secs)	Total Load	Node Count	Model Size (bytes)
1	5	2.45	5	3.01	1.54	3412726	349	48440
2	5	3.02	7	3.04	1.70	2624841	497	55336
5	3	3.43	7	3.06	2.05	1875815	767	62672
10	5	3.46	7	3.09	2.36	1424922	1067	70356
20	9	3.55	7	3.11	2.70	1132646	1479	80928
50	19	3.86	15	3.21	3.32	886360	2297	103784
100	29	3.87	27	3.36	3.99	725231	3145	124920
200	37	3.90	29	3.47	4.68	605193	4379	157268
500	93	4.48	31	3.60	6.44	507056	6995	227972
1000	183	5.12	75	3.84	8.37	462269	10001	309112
2000	367	6.11	87	4.02	11.04	428110	14503	430416
5000	915	7.47	227	4.34	19.82	393041	24867	709560
10000	1831	8.97	281	4.79	34.98	375255	37769	1058268
20000	3661	10.49	281	4.98	73.92	361368	57139	1574172

Table 7.1 Division statistics for simple tool models

(regular division)

α	Max. (Write)	Ave.	Max. (Behead)	Ave.	Time (secs)	Total Load	Node Count	Model Size (bytes)
Best of three								
1	3	2.19	3	3.00	3.48	2933068	349	44464
2	3	2.40	3	3.00	3.95	2083423	469	48328
5	5	2.67	7	3.02	4.57	1440546	709	56296
10	7	2.75	7	3.03	5.23	1113732	1011	65876
20	5	2.88	11	3.06	5.97	827616	1487	79060
50	11	2.66	11	3.07	7.17	578711	2419	102100
100	21	2.65	11	3.10	8.36	468833	3361	125320
200	39	2.73	15	3.13	9.91	393569	4805	163204
500	41	2.94	53	3.26	13.05	327664	8061	248688
1000	79	3.05	69	3.43	16.97	285970	12533	365772
2000	159	3.24	69	3.51	23.08	255472	19023	534160
5000	395	4.03	95	3.57	34.43	235317	28891	788364
10000	791	5.14	247	3.64	49.59	227080	38575	1037952
20000	1579	7.08	247	3.70	78.36	222020	50427	1347976
Best of nine								
1	3	2.40	3	3.00	7.54	1604568	187	35488
2	3	3.05	3	3.00	8.08	1282508	255	38056
5	7	3.75	5	3.01	9.34	982943	405	43408
10	7	3.68	11	3.04	10.47	766987	633	50164
20	11	3.56	13	3.08	12.10	599182	963	59092
50	9	3.30	23	3.22	14.80	436735	1659	78284
100	15	3.10	23	3.21	17.14	354733	2431	99096
200	31	3.05	27	3.27	20.42	294983	3645	131188
500	75	3.25	27	3.32	26.91	240302	6061	195632
1000	147	3.33	69	3.42	34.69	211270	9063	274336
2000	213	3.79	277	3.63	45.56	194146	12809	372528
5000	407	4.71	277	3.58	68.99	182851	19763	553744
10000	813	5.96	277	3.64	99.84	178106	26847	738180
20000	1625	7.81	277	3.76	156.68	174793	36665	993884
Best of twenty-one								
1	3	3.50	3	3.00	17.23	1764728	177	36708
2	5	3.68	3	3.00	19.24	1343670	251	38740
5	5	3.45	9	3.06	22.61	911122	421	43680
10	7	3.55	9	3.04	25.74	744185	653	50360
20	5	3.40	13	3.14	29.76	558002	1021	61532
50	11	3.14	13	3.15	35.44	391232	1703	79648
100	13	3.04	13	3.15	40.71	308425	2447	98912
200	23	3.27	13	3.13	45.93	264450	3271	120684
500	57	3.48	39	3.26	58.20	215972	5217	172416
1000	111	3.78	49	3.38	71.88	193268	7401	230312
2000	221	4.35	67	3.50	90.92	178435	10287	307192
5000	373	5.58	241	3.62	133.97	168851	15423	442072

Table 7.2 Division statistics for simple tool models (with different division strategies)

α	Max. (Write)	Ave.	Max. (Behead)	Ave.	Time (secs)	Total Load	Node Count	Model Size (bytes)
1	3	2.39	5	3.01	3.92	3908772	435	107780
2	3	3.14	5	3.01	4.14	3097813	599	114480
5	5	3.20	5	3.04	4.58	2099940	897	121156
10	7	3.24	5	3.07	4.91	1508255	1249	129156
20	9	3.54	9	3.09	5.40	1202681	1677	140408
50	19	3.64	15	3.17	5.99	919370	2513	161420
100	19	3.69	27	3.31	6.68	750060	3407	183480
200	37	3.91	27	3.41	7.49	641894	4653	216556
500	93	4.41	31	3.59	9.22	530861	7541	295208
1000	183	5.05	75	3.81	11.37	484398	10807	385460
2000	367	5.93	87	3.99	14.78	447591	15719	519136
5000	915	7.08	251	4.43	24.37	407967	27529	841480
10000	1831	8.33	281	4.84	40.95	387237	42143	1241472
20000	3661	9.90	281	4.93	81.32	372564	62991	1799828

Table 7.3 Division statistics for complicated tool models

β	Max. (Write)	Ave.	Max. (Behead)	Ave.	Time (secs)	Total Load	Node Count	Model Size (bytes)
0.00	1831	8.97	281	4.79	35.30	375255	37769	1058268
0.10	1217	8.20	281	4.79	31.01	375362	37681	1055724
0.20	1137	7.64	281	4.81	28.19	375949	37169	1042668
0.50	913	6.61	281	4.81	24.88	376113	37021	1038596
0.75	813	5.94	281	4.79	22.78	376651	36633	1028292
1.00	739	5.56	265	4.63	21.37	377568	35983	1010540
2.00	561	5.01	229	4.25	14.82	408001	24713	706328

Table 7.4 Division statistics for simple tool models (α = 10000)

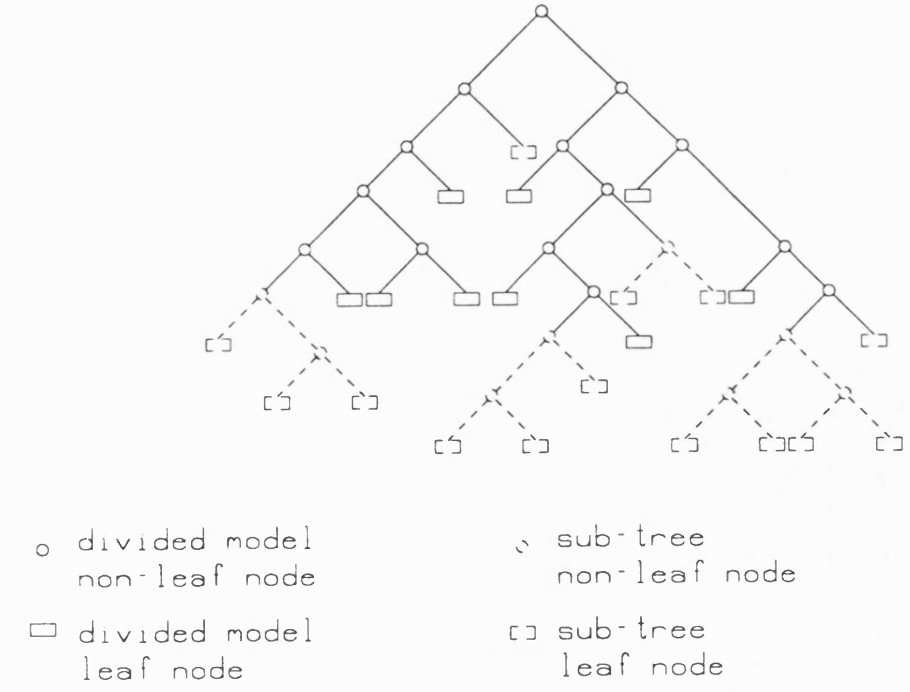

Figure 7.1 The logical tree-structure of the model divider

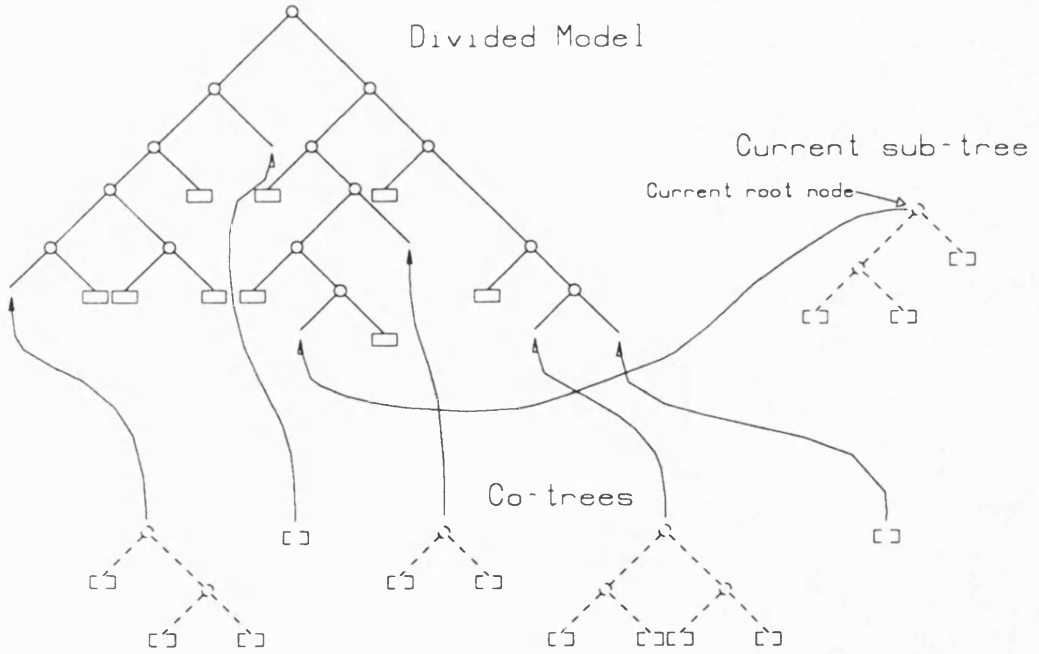

Figure 7.2 The actual tree-structure of the model divider

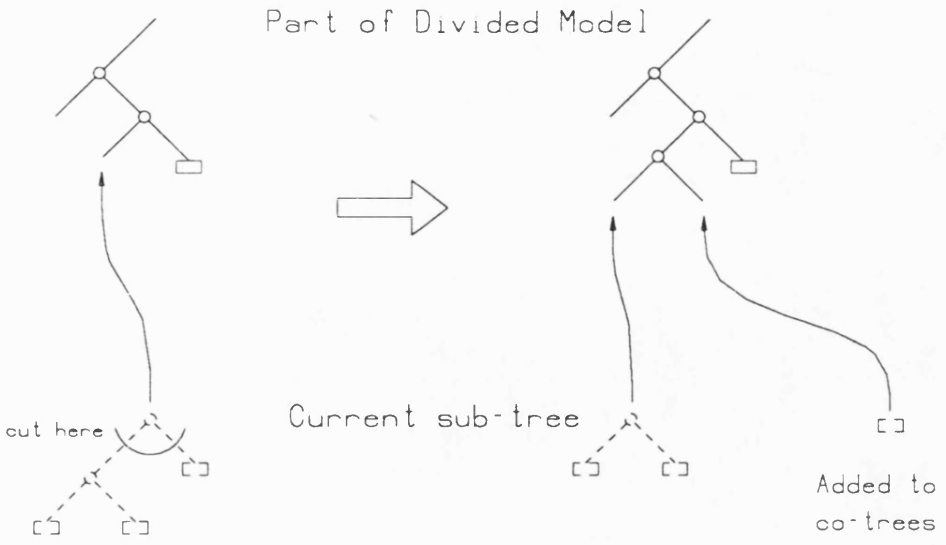

Figure 7.3a Beheading

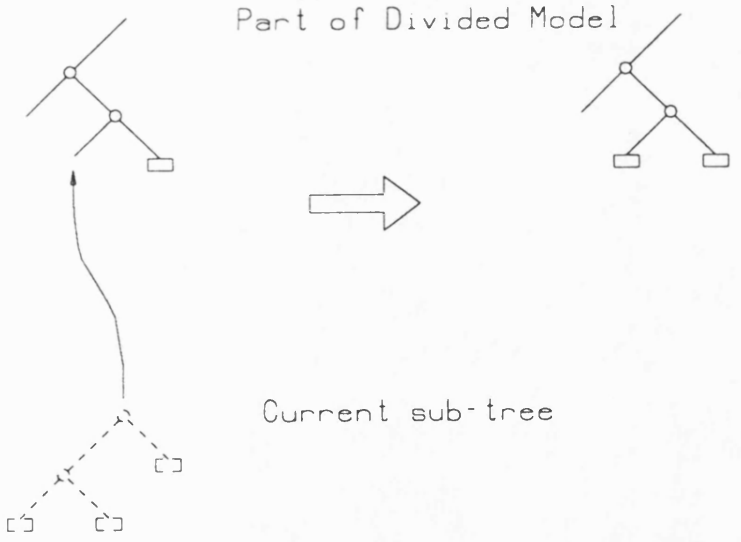

Figure 7.3b Adding to divided model

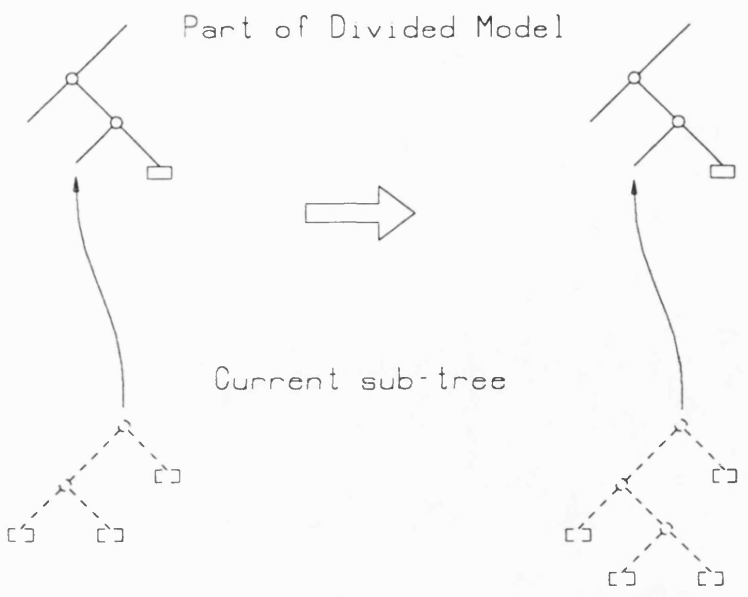

Figure 7.3c Splitting

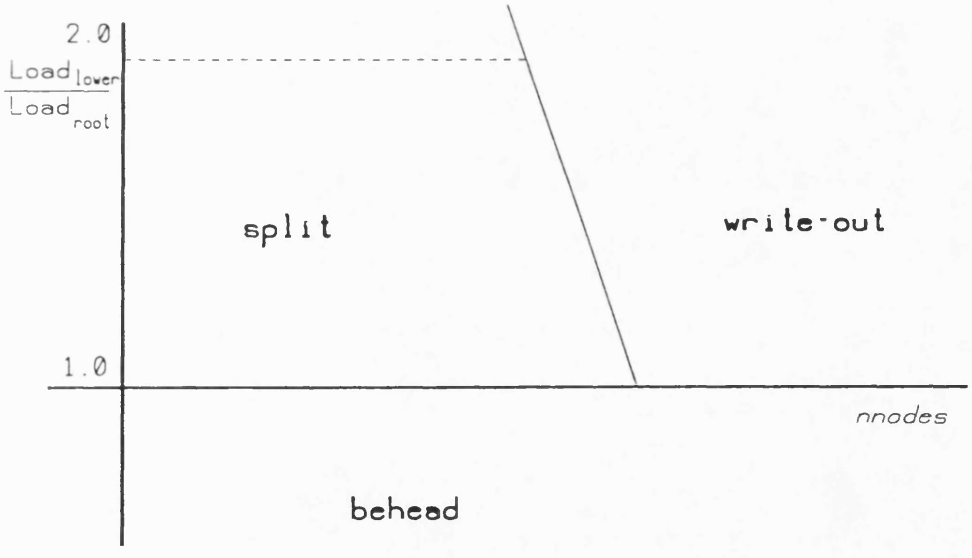

Figure 7.4 Graph of the division process

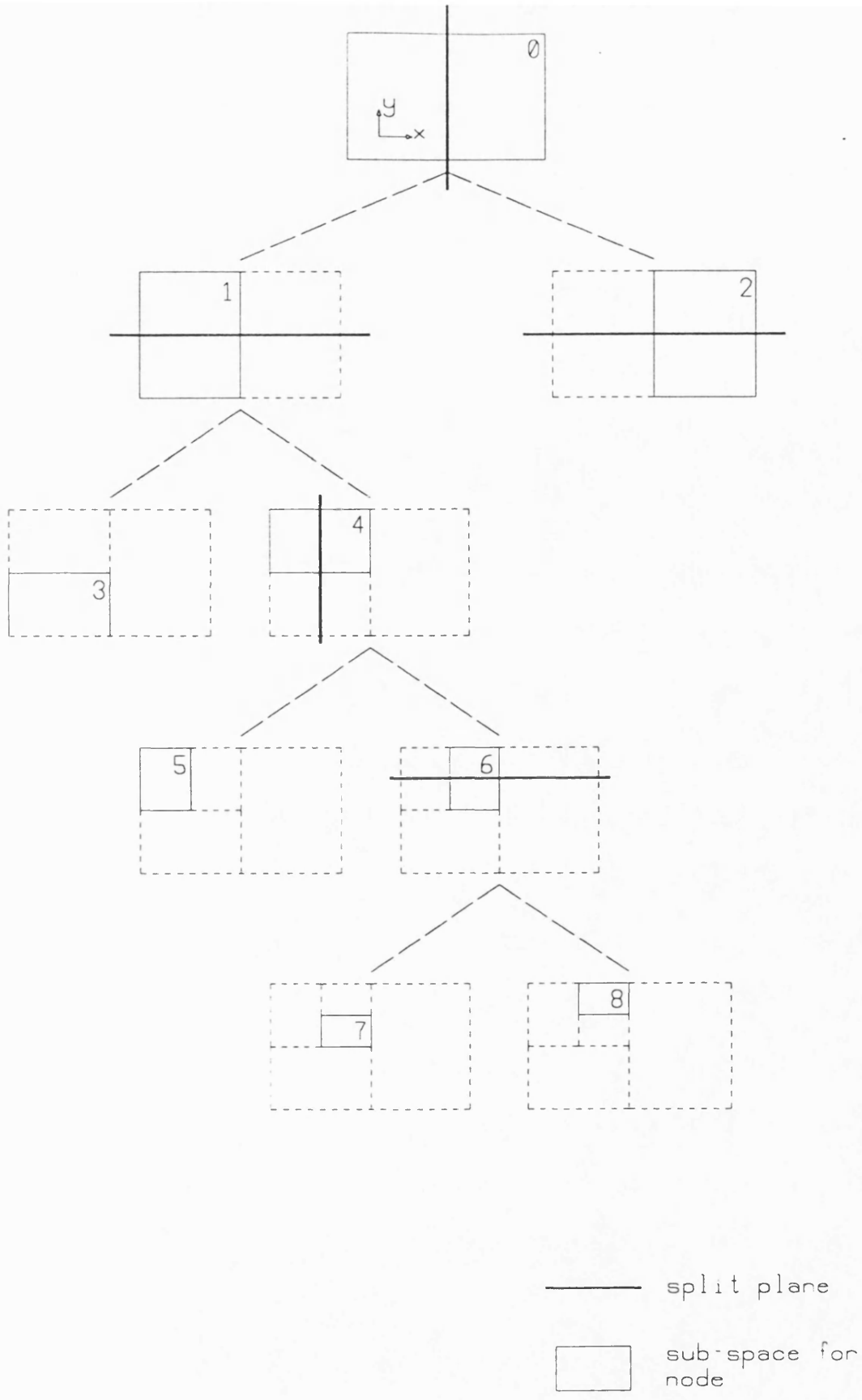

Figure 7.5 The structure of the divided-model tree

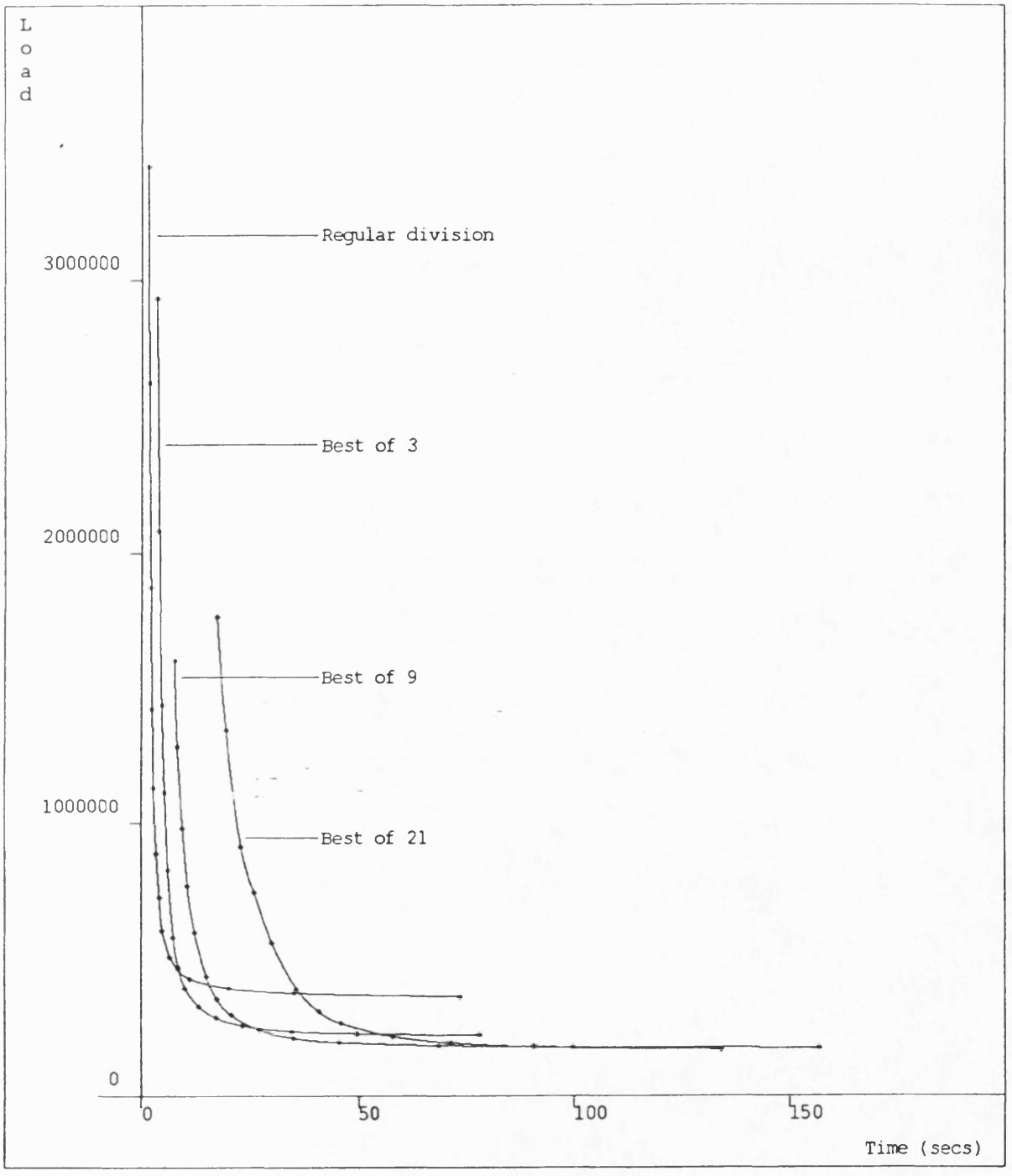

Figure 7.6 Division time vs total (calculated) load

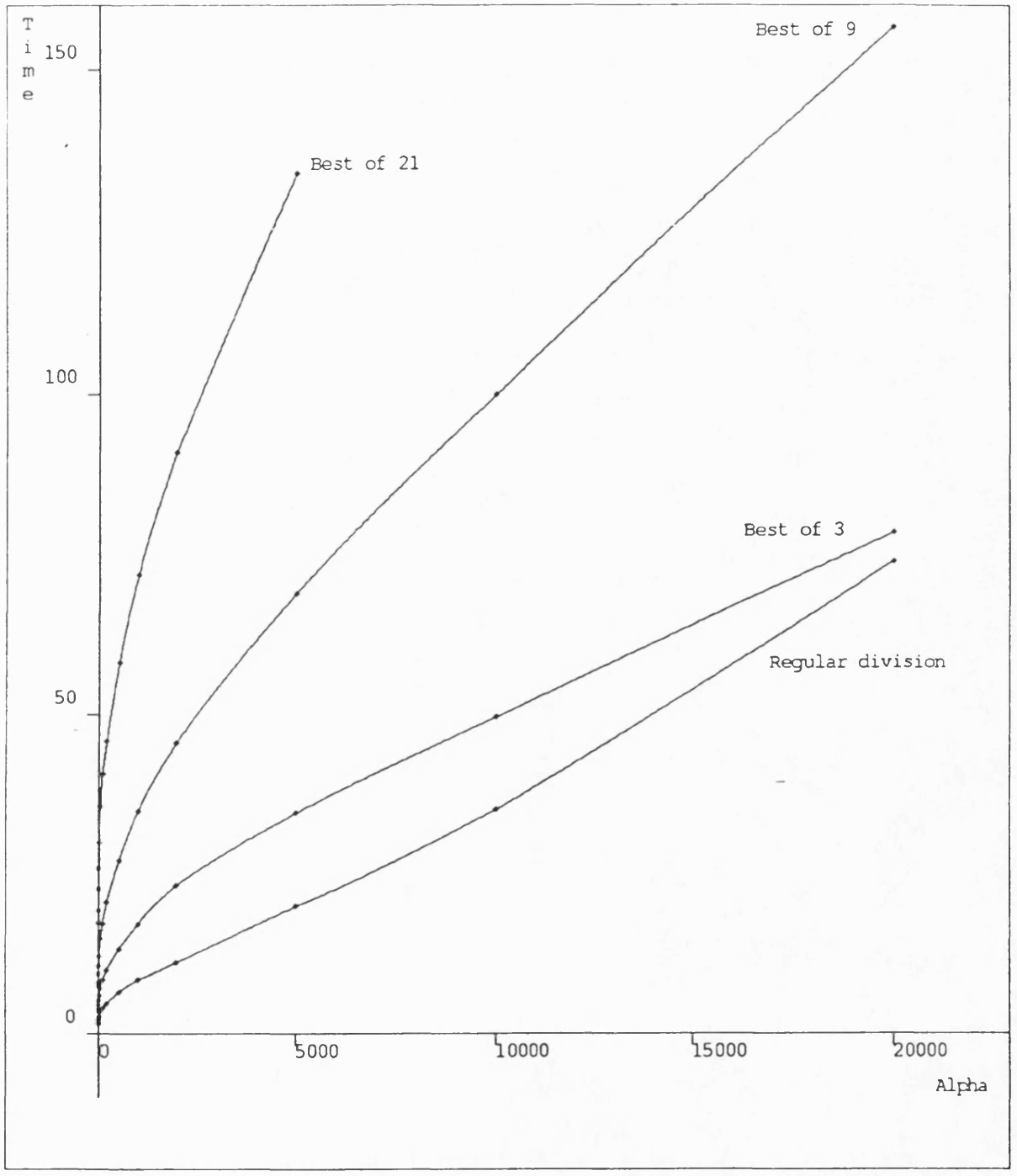

Figure 7.7 Tuning parameter (α) vs division time

CHAPTER 8

Examining the divided model

The third and fourth stages of the toolpath verification system provide facilities for the examination of the model. This is achieved by generating pictures of the divided model, and allowing the user to interactively interrogate the model. These two functions are described separately, although, as will be seen, they share a common technique. The interactive interrogation of solid models is described in [60] both as part of a toolpath verification system, and also as a more widely applicable tool for use with general solid models.

The pictures created from the divided model are displayed as shaded colour images. They are generated by ray-casting. It was decided to use raycasting, as opposed to the other methods of image generation described in Chapter 2, for several reasons. Firstly it is necessary to generate continuous-tone images of the model. This is required since the surfaces generated by machining operations lie at arbitrary orientations, and with complicated components, line images would be confusing. Also, many of the surfaces that are to be modelled are curved. If these are facetted, then the large number of edges will make the picture difficult to interpret, if they are not facetted then, as explained in Chapter 2, they will be difficult to visualise. Ray-casting is a simple, effective method of generating continuous tone images. Secondly, a range of half-space geometries may be easily handled by the raycasting algorithm. The only requirements are firstly that it is possible to find the intersections between the ray vector and the half-space, and secondly that

the surface normal at a specified point on the half-space surface may be calculated. With polynomial half-spaces neither of these presents a problem. Lastly, raycasting may be used both for image generation, and subsequent model interrogation, as described in this chapter.

Picture Generation

Unless the toolpath to be checked is very simple it is probable that not all the regions of interest will be capable of being viewed from a single view-point. Also it may be desirable to view certain parts of the model at a larger scale than others. One or more views of the complete model will almost certainly be required. The raycasting technique described in this chapter allows images to be generated (from the divided model) with the resolution and viewing parameters defined by the user. For each view, the user specifies the viewpoint, center of interest, image resolution and lens angle required. The system allows the user to specify as many sets of these viewing parameters as he requires. Up to eight different pictures may be displayed at the same time on a Sun bit-mapped display. These may be chosen by the user from a larger selection of images as required.

The Raycasting Algorithm

A simple ray-casting technique is described in Chapter 2. In order to use the technique on a spatially divided structure some modification to the basic algorithm are needed. Each ray that is cast will pass through one or more leaf-node sub-spaces in the divided model. Each of these leaf-nodes may be treated using the simple

algorithm, with the additional constraint that ray intersections that lie outside the leaf-node's sub-space are ignored. Hence the additional requirement is to detect which leaf-node sub-spaces are intersected by the ray-vector, in increasing distance along the ray. This addition to the simple algorithm may be regarded as a partial traversal of the leaf-nodes of the tree.

In choosing a method there are two factors to consider. Firstly, only a (very) small proportion of the total number of leaf-node sub-spaces are intersected by a ray. Secondly, depending on the physical positioning of the model, it will often only be necessary to test the first part of the total number of sub-spaces intersected. Hence the method of accessing the sub-spaces should only consider those sub-spaces intersected by the ray, and should generate a list of sub-spaces incrementally in the order required. If this is done then the complete list will not need to be generated except for rays that do not hit the object.

The method for generating the list of sub-spaces is similar to that used to classify which leaf sub-space contains a given point which was described in Chapter 3. The sub-spaces of the two sons of a non-leaf node combine to form the sub-space for that node. Hence if a vector, in this case the ray, passes through the sub-space for a non-leaf node then it must pass through the sub-spaces for either one or both of the sons to that node. Using this property, the tree structure of the divided model can be utilised to generate the sub-space references in the required order.

Details of the tree traversal algorithm and also modifications to the simple membership-test and the handling of polynomial half-spaces are given later in this

chapter.

Colouring the Pictures

The half-spaces are shaded using a simple Lambert cosine law (a lighting vector may be specified by the user). The choice of colours is left to the user. These may be chosen so as to highlight the difference between those surfaces that are part of the original blank, and those which have been produced by a cutter, or so as to generate more realistic pictures. Those surfaces that are generated by rapid tool movements, by non-cutting parts of a tool, or by intersections between parts of the machine-tool and the component may be coloured distinctively, as may 'section' surfaces.

Interrogating the Model

Having generated the images, the final stage in the verification process is to provide the user with adequate facilities so that he may using them to verify that the model is correct. Gross errors in the toolpath, (for example, resulting from specifying the wrong tooling) may be detected visually from the images. Other errors may be undetectable from a visual examination of the model and require detailed geometric information regarding the positioning of surfaces (for example the distance between two surfaces) to be obtained from the model.

Therefore, there is a requirement to display geometric information, extracted from half-spaces in the model, as requested by the user. Only certain geometric information, mainly that which relates to the original component specification, will

be of interest to the user. Rather than try to extract this information, and display it, for example by labeling the surfaces together with a list on the relationships between them (which is not a practical method for any but the simplest of models), it was decided to allow the user to interactively specify the information he needs.

The geometric information relating to a given surface may be obtained from the geometric definition of the half-space that contains it. Other information is usually related to the CLdata, or part-program, block number that contained the cut that generated a given surface. Each half-space is tagged with the block number that generated it. Hence all the required information relating to a part of the component model may be obtained from the half-space records.

A cursor is provided that can be positioned anywhere on the graphics screen. Its position is controlled using a graphics tablet or mouse. In order to indicate a feature that is of interest on the model the cursor is positioned so that it points at the feature in any of the views.

When pointing at a feature on the model in this manner the surface of interest is in fact the first surface that would be hit by a ray originating at the eye position and passing through the pixel containing the cursor. This is the same geometric construction as was used when ray-casting to produce the images. Hence, in order to identify which half-space corresponds to the feature being pointed at, a single ray is cast using the technique already described.

Obviously, the point on the surface hit by the ray lies directly behind the cursor in the image that contains the cursor. It is useful to display this point in the

other pictures on display. To achieve this a ray is constructed, for each of the views, from the view-point to the point on the surface. If this ray reaches the point without first intersecting another real surface then the point will be visible, otherwise the point is not visible. If the point is visible, then a symbol is plotted at the point on the screen corresponding to the ray. The effect of displaying the point in this way is that of a cursor that is moved in space so that it is always on the surface of the model.

Inspection Requirements

Having described the method of identifying surfaces of interest, now consider what facilities are necessary or desirable for processing the information obtained.

The information that may be obtained directly from the model is the (x,y,z) location being pointed at, together with the half-space equation of the surface. In this form it is not easily compared with the relationships between surfaces that are expected. One way of processing the information is to supply the user with a set of routines for manipulating the surface equations (an approach that is used to process the point data obtained from coordinate measuring machines). This still requires a certain amount of data manipulation by the user and does not seem to be particularly satisfactory for much of the inspection requirements in a toolpath verification application.

Geometric information that is of interest for toolpath verification is almost always in the form of relative distances or angles between two points or surfaces. In general, information relating surfaces that are pointed to is more useful than that

relating the actual points, firstly since it is difficult to point accurately, secondly because properly designed engineering components have mainly distances or angles between faces specified, rather than between arbitrary points on those surfaces, and thirdly because, in a machining process, it is the surfaces that are generated. In some cases, other geometrical elements (mainly lines) may be of interest. For example, when dealing with cylindrical surfaces, the central axis of the cylinder is often of interest. Angles and distances between the intersection-lines of surfaces may also be of interest. Point and surface information is obtained directly by ray-casting, whenever information relating to lines is required, so lines are generated either from a single half-space (as in the case of the cylinder centreline), or else from the intersection of two half-spaces).

It is often the case that many of the dimensions of a component are specified relative to a few (planar) datum surfaces. Use is made of this fact, and rather than the user having to specify a pair of surfaces for each measurement, a datum surface may be defined, and then measurements relative to that surfaces obtained by pointing to any surface in the model.

The aim of the interrogation stage of the verification system is to provide a number of 'software measuring tools' that mimic, where appropriate, the physical measuring instruments used in an engineering workshop. The tools provided allow the measurements corresponding to those made with height gauges, internal and external calipers, bore gauges and angle gauges.

Inspection Tools: Pointing

Whenever the user points to a surface, the following information may be displayed:

- the (x,y,z) coordinates of the point specified
- the (x,y,z) coordinates of the point relative to a user-specified origin.
- the number of the block that created the surface.
- the number of the tool that created the surface.
- the source line in the part-program that created the surface.

Depending on the type of surface that the point lies on, additional information may be displayed. If the point lies on a planar surface, then the characteristics of that surface are displayed. For horizontal surfaces the height of the surface is displayed. If the surface is vertical, then the orientation of the surface, relative to the positive x direction, is displayed. For surfaces at arbitrary orientations, the normal vector for the surface is displayed, together with the minimum distance between the surface and the origin. If a datum surface is defined, then the angle between the surface and the datum surface and the distance from the point to the datum surface are displayed, or if the two surfaces are parallel, the distance between the surface and the datum surface is also displayed. If the point lies on a vertical cylindrical surface then the additional information displayed is the x and y coordinates of its centre, together with its radius.

Inspection Tools: Stepping

Having pointed to the surface, the user is then given the choice as to whether to perform further interrogations, based from this first surface point and half-space. If required, a second ray may be generated from this point, normal to the half-space and directed either into or away from the surface (see figure 8.1). Intersections of this ray with the model form the basis for further model interrogations. The user may now step forward, or backward along the ray to any intersection of this second ray with the model. At each step, the same information is generated as for a simple pointing operation, with measurements made relative to a 'user origin' defined as the initial surface point.

If both the first surface, and the new surface are planar, then the distance or angle between the planes is displayed, otherwise, the distance between points is displayed. This tool thus emulates the operation of the engineer's calipers, although with much enhanced usability, since it allows both the distance between opposite facing surfaces, the distance between like facing surfaces, and the angle between any pair of planar surfaces to be measured. It also allows in a single operation, the inspection of features that would otherwise need to be pointed to in two separate views.

Additional Features

In addition to the directly accessible pointing features described above, there are a number of additional features that the user may invoke.

At any stage, the current surface may be defined as a datum surface for future measuring operations. The output may be displayed in either metric or imperial

units. Any surface, together with the actual point pointed at may be stored. Calculations may then be performed on this data. These functions allow the generation of features such as intersection lines of surfaces, of data relating such features. The functions are:

- Generation of the line of intersection of two planar surfaces.
- Generation of the line of intersection between a vertical planar, and a vertical cylindrical surface.
- Calculation of the distance between a line generated above, and a surface (only valid if the line is parallel to the surface).

Traversing the Spatially Divided Model Tree

In the previous chapter, it was explained that each non-leaf node in the divided model stores the position and orientation of the split-plane that separates its two son-nodes (rather than each node storing its sub-space explicitly). This may be used to advantage in the tree-descent stage of the ray-casting procedure. Figure 7.5 shows a two-dimensional spatially divided tree structure. In this example, all of the leaf nodes in the tree are shown as containing air. Figure 8.2 shows a ray passing through the structure; to traverse it, the leaf-nodes numbered 2, 7 and 5 (in figure 7.5) must be visited in order. Figure 8.3 summarises the traversal of the ray though the divided model using the algorithm detailed here.

At each stage in the traversal algorithm, a record is kept of the distance from the eye point along the ray to its intersection with the near-side and far-side of the sub-space for the node under consideration. Traversal is started by calculating the

intersection of the ray with the cuboid object-space, and recording the near and far distances, marked *n0* and *f0* in figure 8.2. The node under consideration is set to be the root-node of the divided model tree.

The position and orientation of the split-plane for the current-node are extracted from the divided model. The distance to the intersection of the ray with split-plane of the current-node (the point *s0* for the root-node) is calculated. This may then be compared with the near and far distances. The result of this comparison indicates whether the ray passes though the nearest of the two son nodes of the current node, the farthest son, or both of them; ie:

If(split_distance < near_distance) *Then*

Ray passes through farthest son only

ElseIf(split_distance > far_distance) *Then*

Ray passes through nearest son only

Else

Ray passes through both sons.

If the ray passes through the nearest son only then the distances need not be updated, the current node is set to be the nearest son. If it passes through the farthest son only then the near distance is set to the split-plane distance and the current node set to be the farthest son. If the ray passes through both sons then the farthest son, together with the far distance are stacked and the current far distance is set to be the split-plane distance. This process is repeated until a leaf-node is reached. If the node contains a sub-model (ie not *air*), then it is processed as explained below. If a real surface intersection is detected, then processing for this

ray is terminated. Otherwise the near distance is set to be the current far distance and a new node and far distance is pulled from the stack. Processing continues until either a real surface intersection is found, or until the stack is empty. In this case, the ray has passed completely though the divided model without hitting the model (as is the case in figure 8.2).

Since sibling nodes are always written such that the sub-space for the right node is more positive than that for the left, the nearest and farthest sons are easily determined from the direction cosines of the ray vector, ie:

If(split_plane is normal to x-axis) *Then*

If(x direction-cosine for ray is positive) *Then*

nearest son = left son

farthest son = right son

Else

nearest son = right son

farthest son = left son

Endif

Endif

(similarly for *y* and *z* direction cosines)

Processing each Sub-model

The intersections of the ray-vector with implicit half-spaces are found by substituting the parametric equation of the ray into the implicit equation of the half-space. This gives a univariate polynomial, the roots of which give the distance along the ray to each intersection. The first stage of processing a sub-model is to generate

the intersections of the ray vector with each half-space in the sub-model. In fact, only those intersections that lie between the near distance for the current sub-space and the far side of the object-space are found. If, during subsequent processing of this ray, the same half-space occurs in another sub-model, then the roots of the polynomial will not have to be re-found.

All roots that lie further along the ray than the near distance for the current sub-space are inserted into an ordered linked-list. Once all the half-spaces for the sub-model have been processed, the roots are checked in order until a root lying beyond the far edge of the sub-space is reached or a real surface is found. This checking is performed using a membership test between the root point and the sub-model. If a real surface is found then the surface normal vector for the half-space on which the point lies is generated by substituting the z, y and z coordinates into the partial differentials of the surface. Otherwise the tree traversal described above is continued

When performing the membership tests, rather than each half-space being classified for each point to be tested, half-spaces are classified once (when the roots are found) with respect to the 'eye' position. Whenever a root is tested and found not to lie on a real surface, the contribution of the half-space that generated the root is inverted (ie *solid* becomes *air* and *air* becomes *solid*).

During picture generation, only root points whose half-space classifications are *air* need to be tested, since those whose half-spaces are currently classified as *solid* are backward-facing. When casting secondary rays for measuring purposes then either points on these backward-facing surfaces, or points on forward-facing half-

spaces can be ignored depending on whether the ray is currently passing through solid or air.

Rootfinding for Polynomial Half-spaces

The roots for planar and quadratic half-spaces are found directly. The roots for half-spaces of higher order are found using an interval-splitting root-isolation method. This starts with an interval along the ray bounded by the point where the ray first hits the sub-space under consideration and that where it leaves the object-space. The interval is recursively split until a *Sturmn sequence* evaluation of the half-space polynomial reveals that a single root has been isolated. The exact root is then found by binary-chopping the interval until the root is found to the required accuracy.

This technique allows roots to be found for polynomials of any degree up to a limit imposed by the numerical stability of the algorithm implemented in floating point arithmetic. For the range of tools shapes and motions capable of being processed by the verification system described in this thesis, a direct method could be used since the polynomial surfaces are all of degree less than five.

The Performance of the Raycaster

Tables 8.1 and 8.2 show the performance of the raycaster for the divided models summarised in tables 7.1 and 7.2. The 'Time' column contains the elapsed time for generating a single picture (one of those displayed on the screen pictures in Appendix 1) with a resolution of 768 by 512 pixels. For all four division stra-

tegies, the time decreases rapidly at first as the amount of division in the divided model increases. This may be attributed to the large reduction in the number of root findings and membership tests required, as shown in the last four columns. Columns 6 and 8 in tables 8.1 and 8.2 show the number of root-finding operations that were actually performed when generating the picture, columns 7 and 9 show the number that where not performed due to the roots already having been found when processing a prior sub-model.

As the size of the divided model grows, the time for each ray becomes nearly constant. This is because the rate of reduction in the number of root findings per ray decreases, and the number of sub-spaces that are traversed by the ray prior to its hitting a real surface in the model increases. Figure 8.4 shows a graph of the ray-tracing times for different values of the division tuning parameter α (α is plotted on a logarithmic scale). The data is taken from tables 8.1 and 8.2.

The total combined times for model division and ray-tracing a single picture are plotted in figure 8.5 (data from tables 7.1, 7.2, 8.1 and 8.2). For small values of α, the total time values are similar to the time for raycasting alone since the time taken for division is very small. As the value of α is increased, the increase in the division time outweighs the decrease in raycasting time, and the total time reaches a minimum, and then increases. The precise amount of division that gives the lowest total time will clearly depend on the total number of rays that are required to generate pictures from the divided model. The relatively flat wide region in the middle of each curve indicates that, to achieve near-minimum combined time, a precise value of α is not required.

The time to process a single ray, whilst interrogating the model, is small enough to allow the process to be performed interactively.

Figure 8.6 shows a graph of the calculated total load of a divided model (from tables 7.1 and 7.2) plotted against the actual time to generate a picture (from tables 8.1 and 8.2). The actual time taken to generate the pictures is not proportional to the calculated loads. This may be due either the load function, as described in Chapter 7 not being an accurate model of the actual load incurred, or because the calculated load takes no account of the time required to traverse the sub-space tree during raycasting. The relationship between calculated load and raycasting time does appear to be approximately linear, and it is likely that a more accurate load function could be derived. If this were done, then it is to be expected that the performance of the model dividers would be improved since the decisions that they make would be based on more accurate information.

α	Time (secs)	Sub-spaces hit			Root Solvings (Degree 1)		Root Solvings (Degree 2)	
		Non-leaf	Air	Surface				
1	640.99	15.98	4.61	1.00	4907417	1701726	919046	295158
2	476.60	16.67	4.74	1.00	3508760	1369712	574989	219243
5	359.40	17.39	5.01	0.95	2368661	990963	340460	141115
10	286.98	18.04	5.26	0.87	1534938	675840	205868	97345
20	254.05	18.58	5.47	0.81	1095759	517475	132841	76778
50	234.34	19.35	5.76	0.71	767073	373237	101866	64890
100	227.96	19.75	5.89	0.68	660147	288419	85449	41251
200	219.84	20.12	6.00	0.64	561878	220197	65336	24843
500	213.12	20.70	6.17	0.60	444808	180561	48842	19860
1000	209.97	21.18	6.25	0.60	390028	168662	41548	16791
2000	208.40	21.46	6.30	0.60	362636	161635	35766	13767
5000	207.68	21.98	6.42	0.60	320181	151348	26911	12101
10000	205.34	22.31	6.48	0.60	293646	150947	23302	10943
20000	206.96	22.64	6.55	0.60	272926	146792	21661	10486

Table 8.1 Raycasting statistics for simple tool models (regular division)

α	Time (secs)	Sub-spaces hit			Root Solvings (Degree 1)		Root Solvings (Degree 2)	
		Non-lear	Air	Surface				
Best of three								
1	510.38	15.82	4.54	1.32	4334925	1368692	752288	151435
2	383.51	16.82	4.92	1.06	2787394	913149	452275	127642
5	294.90	17.77	5.25	0.86	1676330	623059	238074	88083
10	260.15	18.39	5.50	0.72	1172577	438901	164082	73565
20	234.71	18.96	5.69	0.63	813936	289165	109718	48262
50	219.18	19.64	5.93	0.55	577488	177122	64618	18283
100	212.26	20.04	6.06	0.51	481144	133859	47648	12538
200	209.29	20.45	6.20	0.48	406381	99778	38875	9520
500	205.05	21.12	6.42	0.47	316027	75155	31579	9165
1000	201.92	21.68	6.60	0.46	238238	57902	22954	9565
2000	199.63	22.05	6.72	0.44	196171	43952	16684	7955
5000	199.23	22.30	6.80	0.43	170734	36048	13514	6883
10000	197.63	22.46	6.85	0.42	158167	31767	11422	6399
20000	198.27	22.52	6.86	0.42	153502	30450	10466	5952
Best of nine								
1	332.06	10.72	3.11	0.57	2243207	279954	429158	34226
2	277.92	11.10	3.15	0.58	1713271	316101	262082	33560
5	243.17	11.65	3.26	0.55	1276983	289996	156340	28798
10	222.07	12.19	3.41	0.52	969291	233474	110132	21550
20	207.43	12.61	3.50	0.51	765519	196312	76100	17296
50	192.07	13.26	3.68	0.48	527782	130335	48492	12025
100	187.39	13.68	3.80	0.46	415077	92452	39068	9665
200	182.17	14.21	3.96	0.44	315043	67157	29056	9882
500	179.68	14.68	4.11	0.42	227692	42436	21461	6924
1000	175.95	14.96	4.21	0.40	190912	31844	15844	5409
2000	175.71	15.12	4.26	0.39	171016	24267	13504	4414
5000	176.46	15.26	4.31	0.38	158501	20694	11758	3725
10000	175.12	15.36	4.34	0.38	152137	19694	10840	3541
20000	175.34	15.44	4.36	0.38	146921	18913	10314	3318
Best of twenty-one								
1	370.09	7.53	2.39	0.54	2825674	375284	436890	34822
2	306.99	7.85	2.48	0.53	2122599	357693	310674	37650
5	230.78	8.47	2.69	0.49	1223437	256702	154428	37230
10	211.56	8.95	2.76	0.49	948643	236383	110055	34893
20	194.62	9.37	2.83	0.49	718722	178724	72331	27997
50	178.92	9.96	2.96	0.47	475873	99116	45012	14085
100	173.21	10.30	3.06	0.44	362218	64339	32645	11319
200	169.54	10.59	3.13	0.42	282282	48674	25717	9657
500	166.44	10.99	3.27	0.39	211433	30844	20056	5481
1000	164.94	11.19	3.34	0.38	185027	21990	15523	3987
2000	163.76	11.33	3.38	0.37	166481	18322	12741	3124
5000	163.28	11.45	3.42	0.37	154784	15800	11440	2558

Table 8.2 Raycasting statistics for simple tool models (with different division strategies)

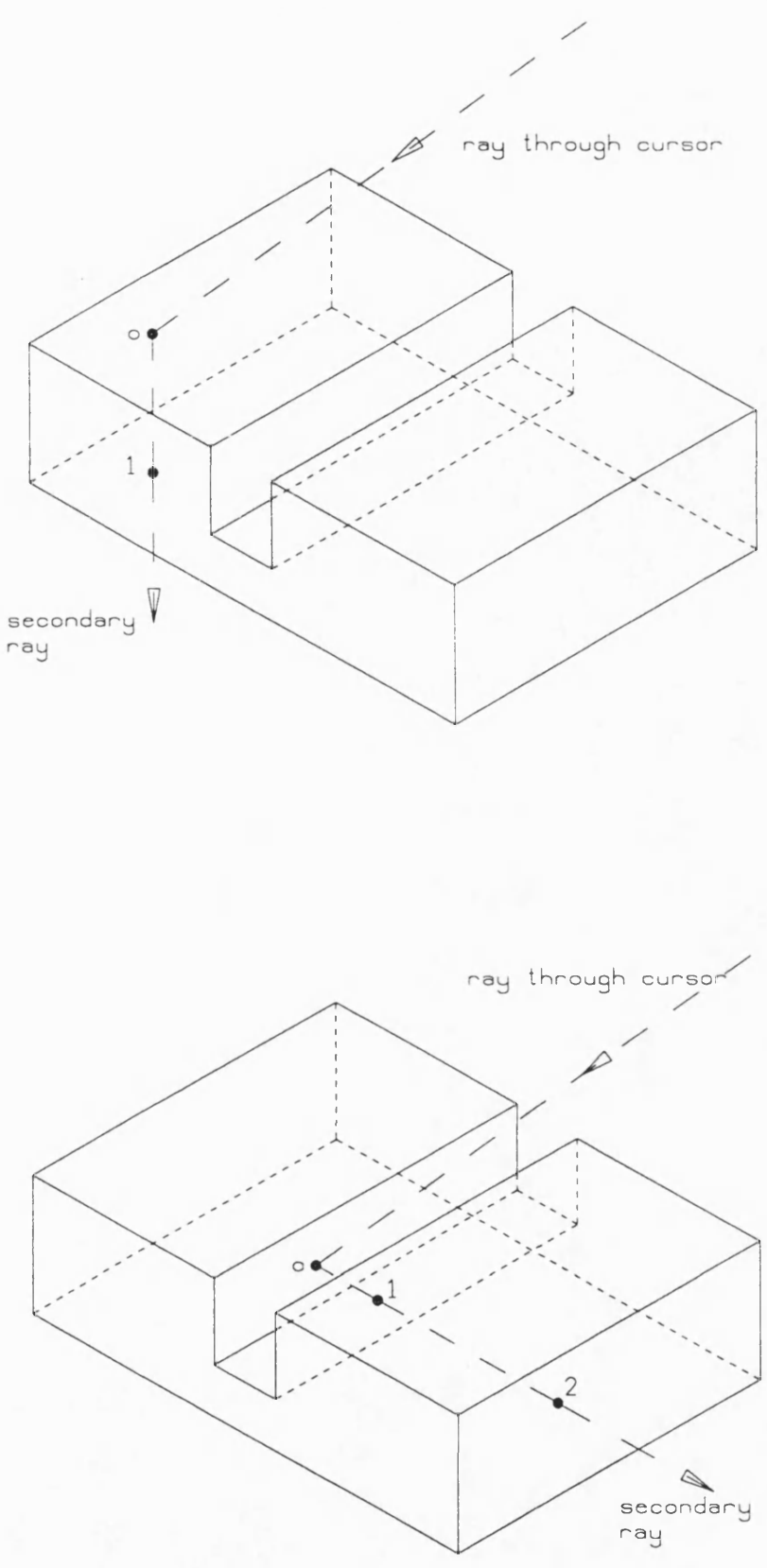

Figure 8.1 Generating Secondary Rays

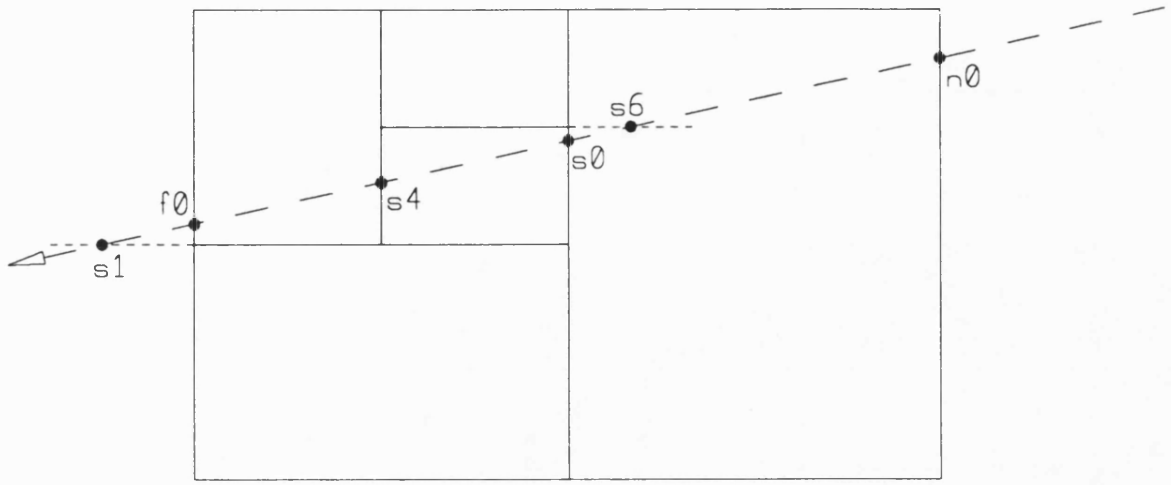

Figure 8.2 A ray passing through a Spatially-divided Model

Sub-space	Limits		Test result			Action
	Near	Far	split < near	near < split < far	far < split	
0	n0	f0		*		stack far son and split dist.
2	n0	s0	leaf node			process sub-model
Pull node from stack						
1	s0	f0			*	node becomes near son
4	s0	f0		*		stack far son and split dist.
6	s0	s4	*			node becomes far son
7	s0	s4	leaf node			process sub-model
Pull node from stack						
5	s4	f0	leaf node			process sub-model

Figure 8.3 Summary of the Traversal Algorithm

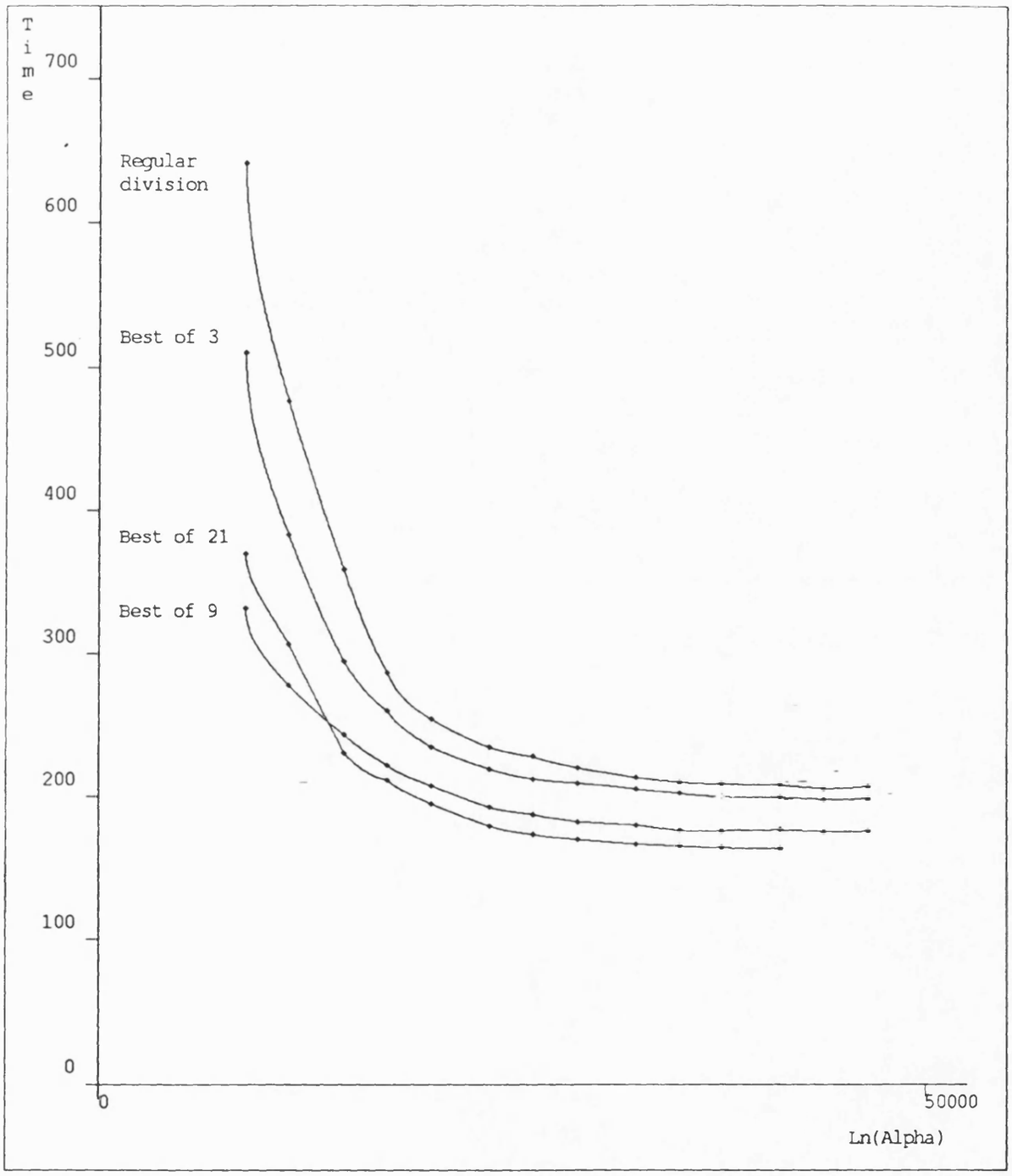

Figure 8.4 Tuning parameter (α) vs raytracing time

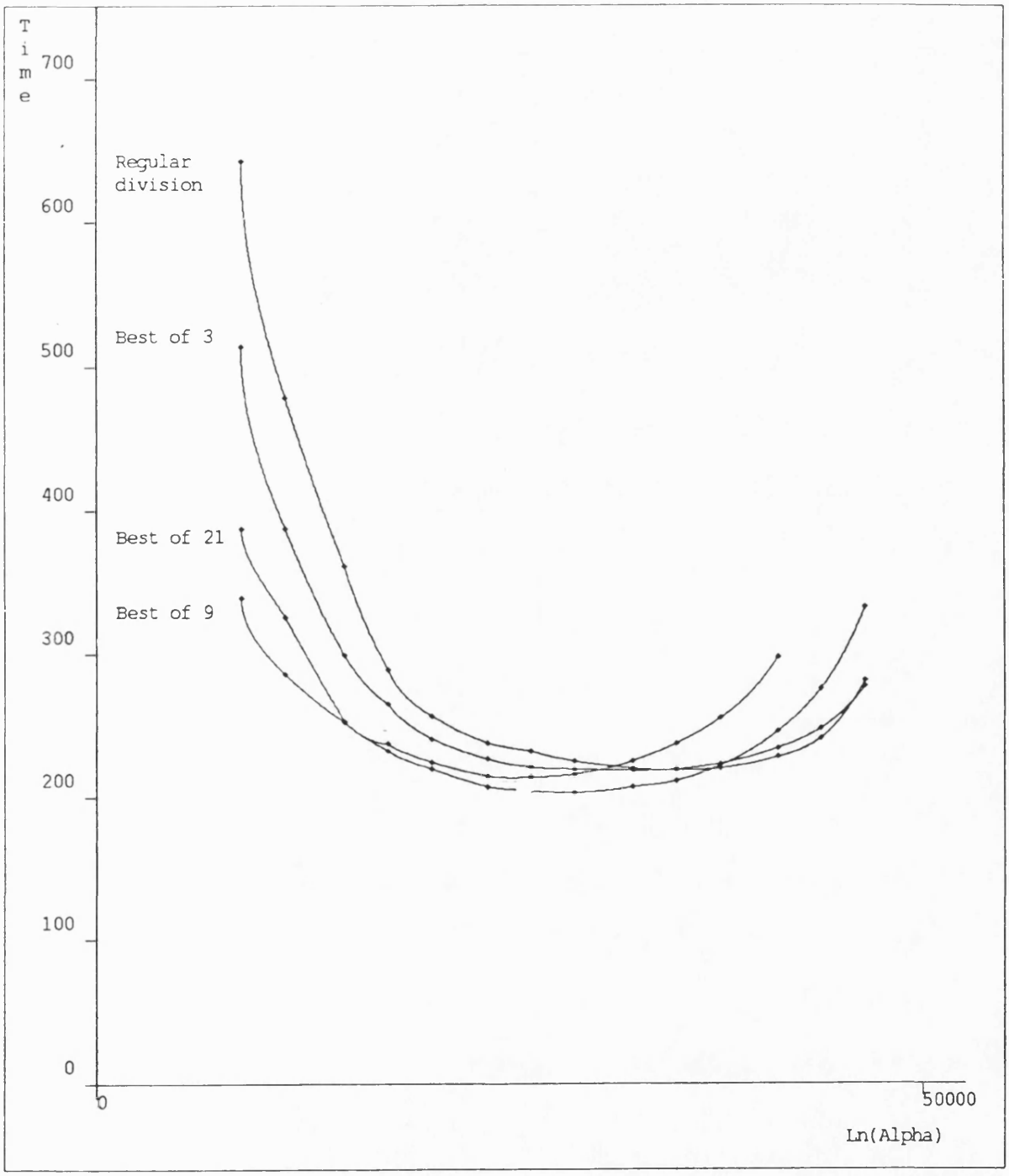

Figure 8.5 Tuning parameter (α) vs combined times

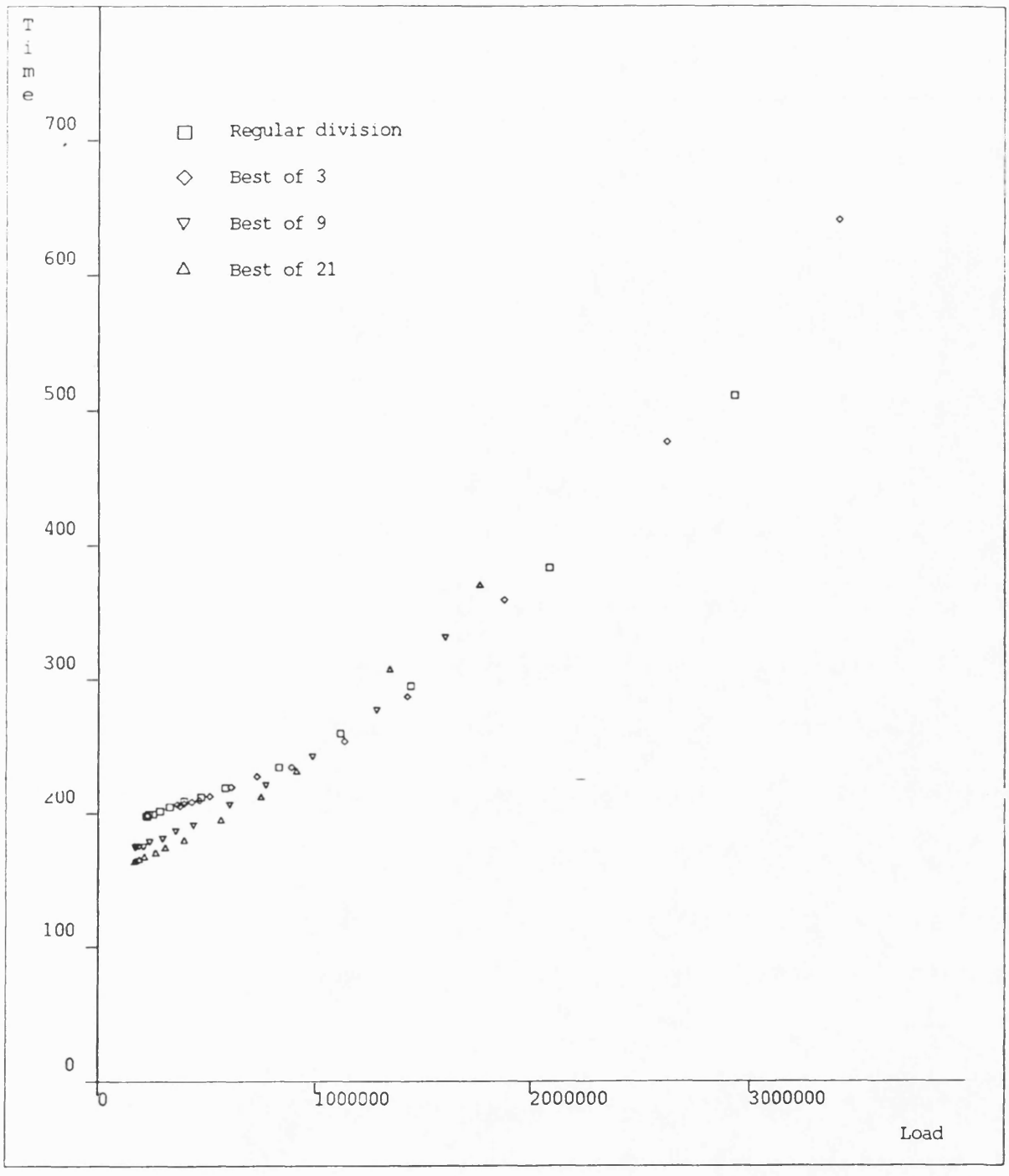

Figure 8.6 Raytracing time vs total calculated load

CHAPTER 9

Conclusions

The Verification System

Numerically controlled machine-tools are in widespread use in the engineering industry. The checking of toolpaths for such machines is of importance if expensive machining errors are to be avoided. The work described in this thesis has shown that a toolpath verification system based on solid modelling techniques is capable of performing such checking.

The verification system has been tested on a range of toolpaths of sizes that are representative of those in industrial use. It has proved suitable for detecting geometric errors that would occur in the machined component that would result from, for example, an incorrect toolpath or the specification of incorrect tooling. Collisions between parts of the machine-tool and the workpiece may also be checked for. Because the system uses human inspection of the automatically generated component model, features such as thin webs (which are not errors as such) that may lead to practical problems during machining may be detected.

In addition to being used for the generation of pictures of the proposed component, and the checking of its dimensions, the solid model of the component may also be used to calculate the volume of model, or the volume of metal removed during the machining process.

Interactive Interrogation of Solid Models

In addition to their use in checking models generated by the toolpath verification system, the model interrogation tools based on ray-tracing have been found to be useful for checking models generated by other means. The ability to check for dimensional errors in models generated using language input has been found to be particularly valuable.

The Solid Modelling Scheme using Spatial Division

The solid modelling scheme used to represent the components that would result from machining - a set-theoretic modeller based on implicit polynomial primitives - has been found to be capable of representing a wide range of useful shapes. A spatial division strategy which incorporates trial division as a technique for controlling division has been shown to be capable of reducing the computational requirements for the processing of large set-theoretic models to a level suitable for current computers. One of the major problems associated with the use of spatial division strategies to reduce the computational loads of solid modellers is how to decide what level of division will enable the most efficient use of the divided model. The technique developed makes the decision based on a knowledge of the potential result of further division.

The level of model division may be tuned to provide a balance between the time taken by the model divider, and that for the subsequent raycasting stage. The spatially divided model provides a basis for much future work as outlined below.

Graphical Input Techniques

An algorithm is described for the input of models described by two-dimensional contours. Although more efficient algorithms have been developed elsewhere, that described in this thesis has the advantage of simplicity.

Future Work

Extending the System

Although the system implemented is restricted to the checking toolpaths for vertical milling machines, the solid modelling scheme used is clearly extendible to perform the verification task on toolpaths for other machine-tools. The only changes that would be required are to the swept-volume model generators. If four and five axis toolpaths are to be checked then the measuring tools provided at the model interrogation stage may need to be altered.

Incremental Cutting Simulation

The verification system described in the thesis is suitable for checking complete toolpaths (it could, of course, be used to check partial toolpaths). Its usefulness for toolpath verification would be enhanced it the toolpath could be checked on an incremental block-by-block basis. This could be done if models for each tool movement were processed individually. The time taken to regenerate pictures of the component could be minimised by making use of the spatially divided structure. The model for each tool movement is incorporated into the divided model, with additional division taking place if required. The region of each picture that

will need to be updated is bounded by the projected outline of the volume that is both swept by the cutter, and also lies within sub-spaces in the divided model whose sub-models have altered. As new surfaces are machined, some parts of the model will be simplified. This will result in some sub-spaces in the spatially divided model becoming *air*, which will reduce the computational load required to process future tool motions.

Multi-processor Implementation

As with any model rendering strategy based on raycasting, the picture generation stage of the verification system is suitable for a multi-processor implementation. The time taken for model division may also be reduced using multiple processors.

Detecting Collisions

The automatic detection of collisions between non-cutting parts of the tool and the workpiece for an individual cutter motion require a 'null object test' to be performed between the volume swept by those non-cutting parts of the tool, and the workpiece model that results from the cut. It will usually be the case that only a small proportion of the workpiece model will be in the locality of the swept volume. The spatially divided model may be used to restrict the null object test to those regions of the model, resulting in a significant time saving.

APPENDIX 1

Examples of the Toolpath Verification System in Use

Figures A1.2 to A1.7 show the interrogation system in use on two components. The first is the shape whose toolpath is shown in figure A1.1. The actions that the figures depict are :

Figure A1.2: The verification system showing four views of the component, a command menu at the lower left corner of the screen and an output window in the lower right corner.

Figure A1.3: Pointing at a feature on the model, in this case the hole in the middle of the top-left view, results in the block number containing the tool motion that created the hole and tool number being displayed in the output window. The source code of the block, and the details of the surface being pointed at are shown in the partially hidden text window in the lower left corner of the screen.

Figure A1.4: The point on the surface of the model lying behind the cursor in the top-left window is now being displayed in the other windows, (although in practice the point is not visible in those views).

Figure A1.5: Having selected the *Step* and *Out* functions, a secondary ray has been generated from the previous surface point, in a direction out from the surface. This has resulted in a second point being displayed (with a diagonal cross), together with the distance between the points also being shown.

The next 2 pictures show a more complicated component, a drawing of which is shown in figure 5.5, and for which a toolpath centre-line plot is shown in figure 5.6. Figure A1.6 shows the width of the web being checked. Figure A1.7 shows the effect of a large number of small tool movements used to create a three-dimensional surface.

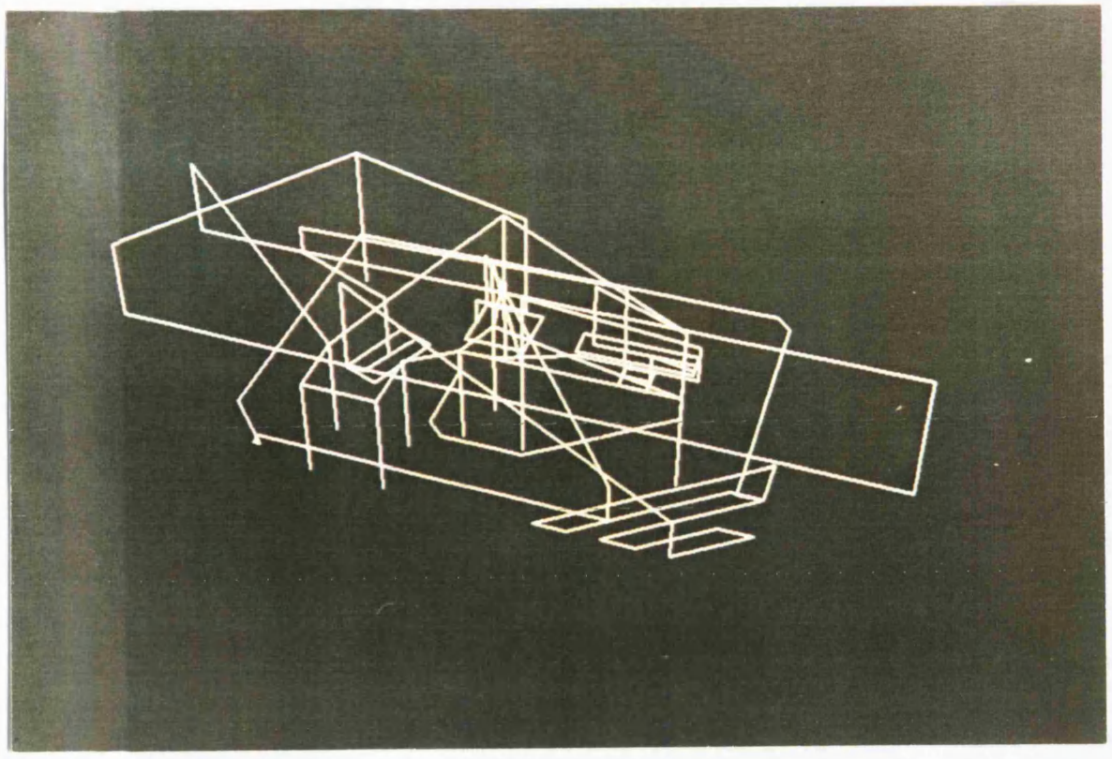

Figure A1.1 Toolpath Centreline Plot

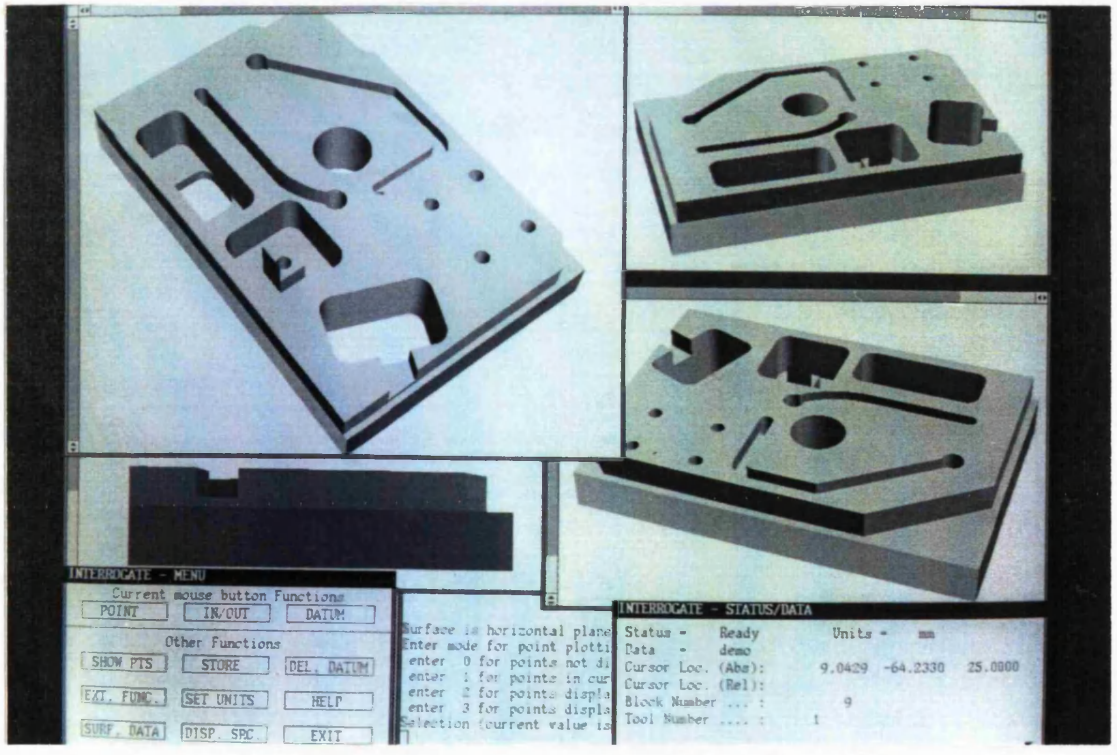

Figure A1.2 The Verification System

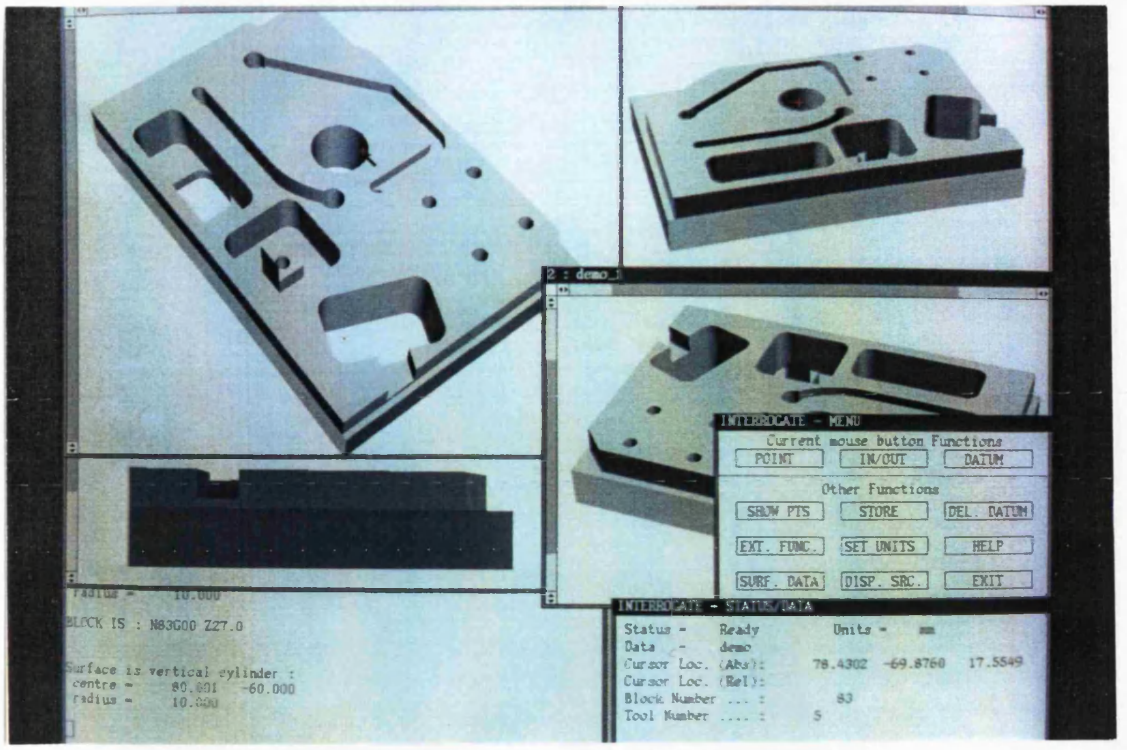

Figure A1.3 Pointing

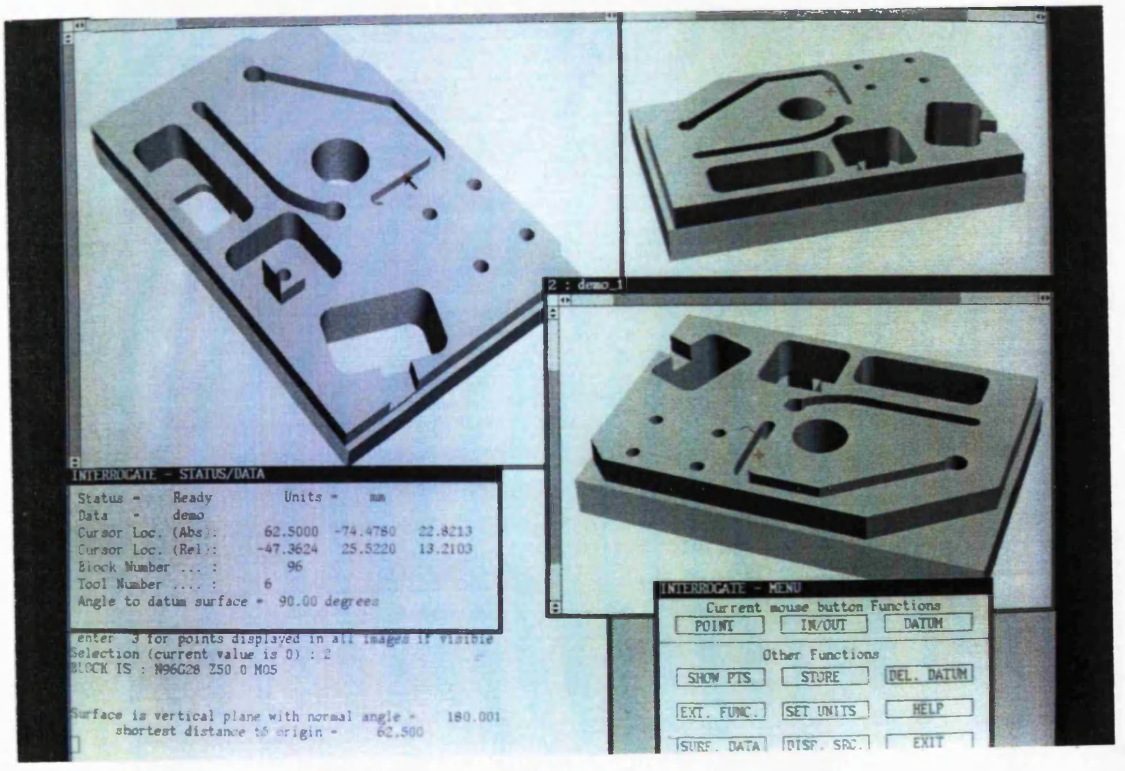

Figure A1.4 Cursor Displays Surface Point

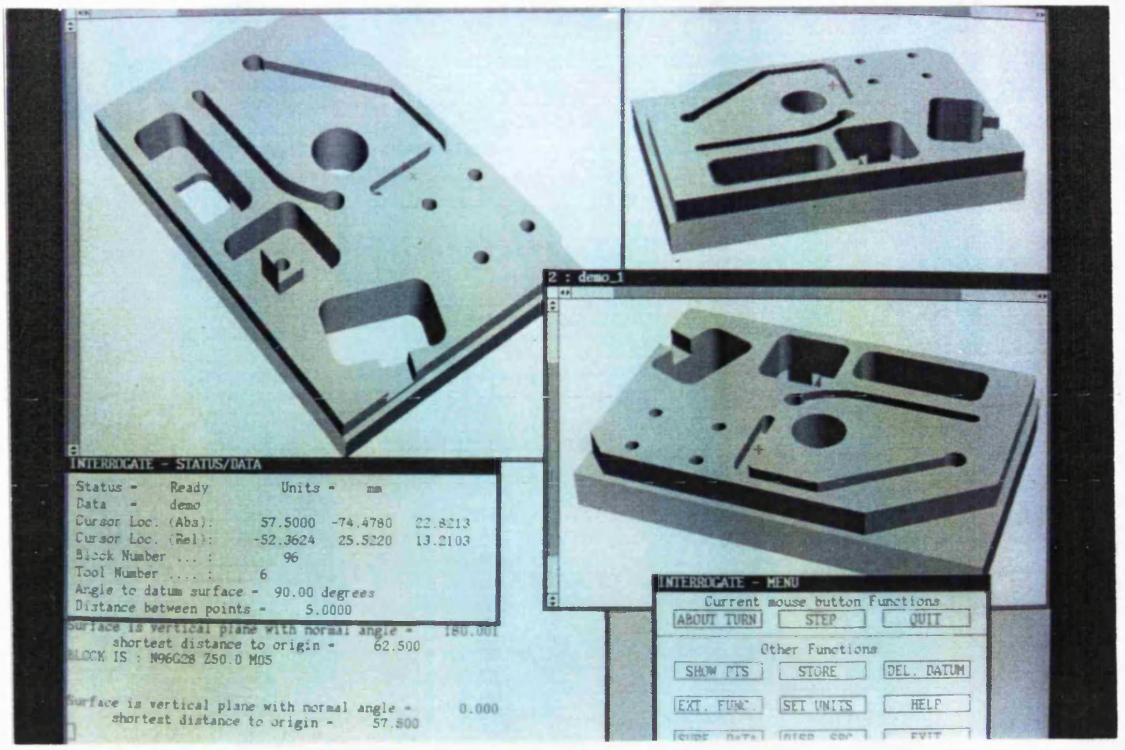

Figure A1.5 Measuring with Secondary Ray

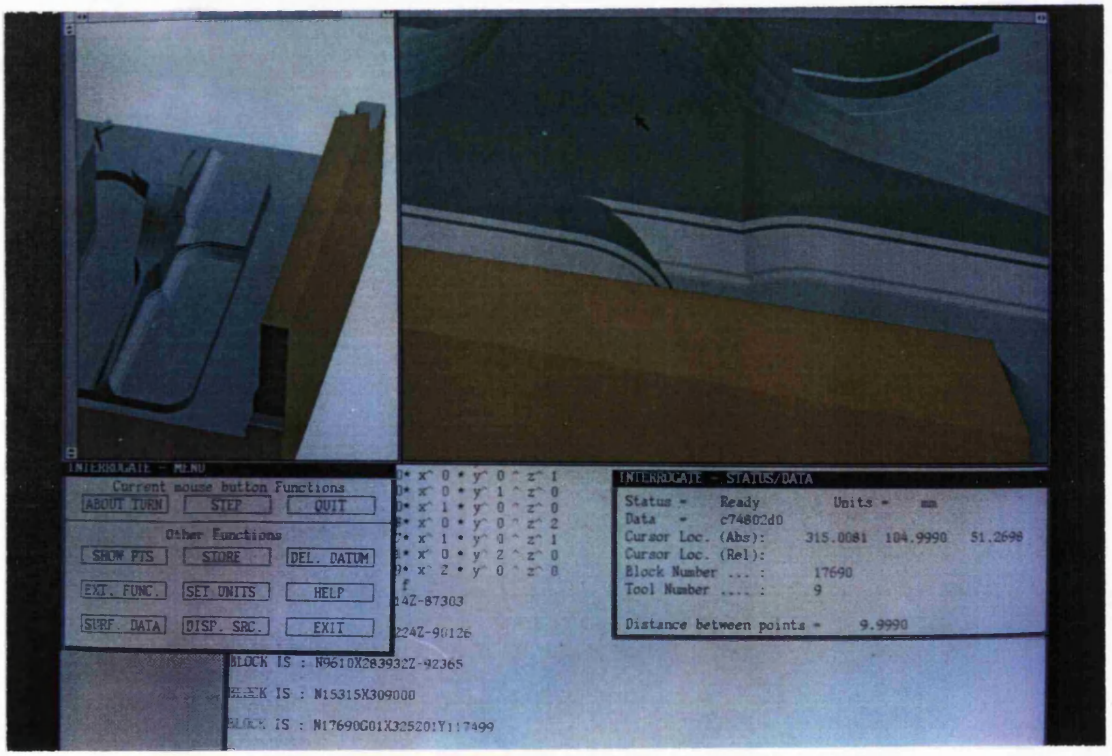

Figure A1.6 A Complicated Component: Measuring

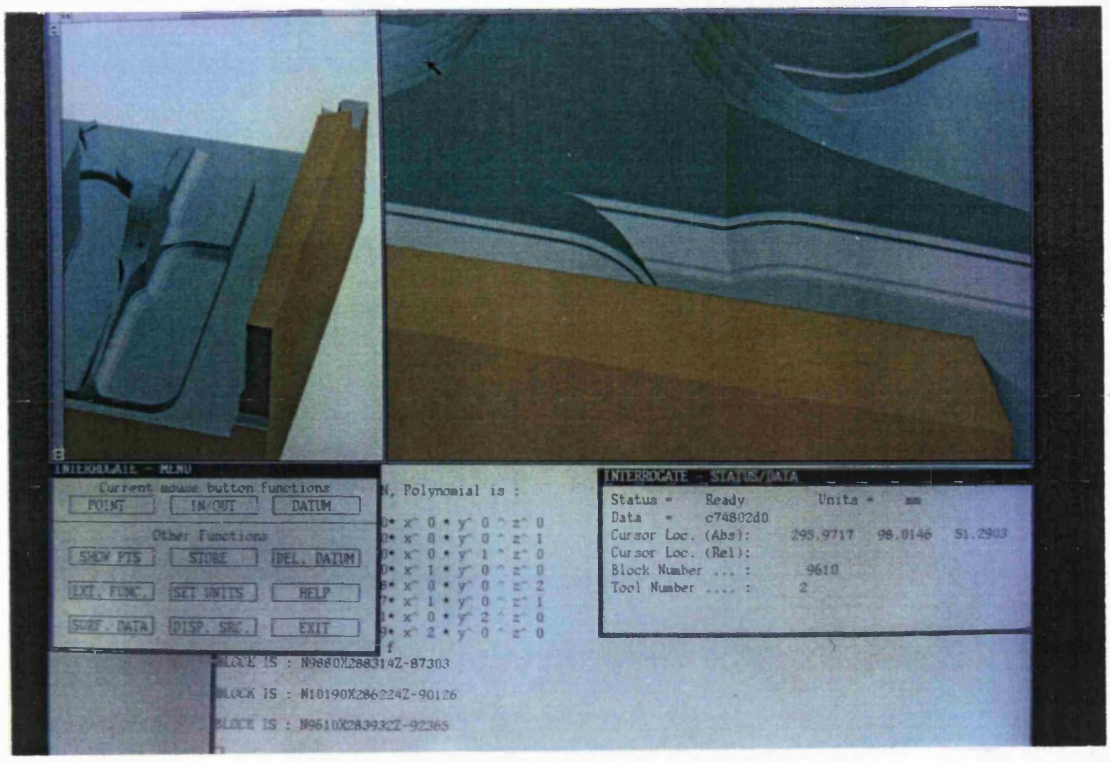

Figure A1.7 A Complicated Component: 3-axis cuts

References

1. J R Woodwark, *Computing Shape*, Butterworths, 1986.

2. I D Faux and M J Pratt, *Computational Geometry for Design and Manufacture*, Ellis Horwood, 1979.

3. A Kela, H B Voelcker, and J A Goldak, *Automatic Generation of Hierarchical, Spatially Addressible Finite Element Meshes from CSG Representations of Solids*, p. Proc. Int. Conf. on Accuracy Estimates and Adaptive Refinements in Finite Element Computation, Lisbon, Portugal, February 1984.

4. T W Stacey and A E Middleditch, "The Geometry of Machining for Computer Aided Manufacture," *Robotica*, vol. 4, pp. 83-91, 1986.

5. J E Bobrow, "NC machine toolpath generation from CSG part representations," *CAD*, vol. 17, no. 2, pp. 69-76.

6. A R Grayer, *The Automatic Production of Machined Components starting from a Stored geometric Description*, Shape data Ltd, Cambridge, England.

7. Dayong Zhang, "CSG Solid Modelling and Automatic NC Machining of Blend Surfaces," *PhD Thesis, University of Bath*, 1986.

8. A A Requicha, "Representations for Rigid Solids: Theory, Methods and Systems'," *ACM Computing Surveys*, vol. 12, no. 4, pp. 437-464, December 1980.

9. B G Baumgart, "Geometric Modelling for Computer Vision," *Stanford AI Lab. Report*, no. STAN-CS-74-463.

10. A Ricci, "A Constructive Geometry for Computer Graphics," *The Computer Journal*, vol. 16, no. 2, February 1972.

11. A A Requicha and S C Chan, "Representation of Geometric Features, Tolerances, and Attributes in Solid Modelers Based on Constructive Geometry," *IEEE Journal of Robotics and Automation*, vol. RA-2, no. 3, September 1986.

12. A A Requicha, "Mathematical Models of Rigid Solid Objects," *Production Automation Project Tech. Mem. 28*, The University of Rochester, Rochester, New York, 1977.

13. B A Leatham-Jones, *Introduction to Computer Numerical Control*, Pitman/Wiley, 1986.

14. A F Wallis and J R Woodwark, "A Set-theoretic Solid Modelling System based on Implicit Blends," *Presented at Theory and Practice of Geometric Modelling Conference, Tubingen, Germany*, October 1988. (Available from A F Wallis, University of Bath, U.K.)

15. R B Tilove, "Exploiting Spatial and Structural Locality in Geometric Modelling," *Production Automation Project Tech. Mem. 38*, The University of Rochester, Rochester, New York, October 1981.

16. I E Sutherland, R F Sproull, and R A Schumaker, "A Characterization of Ten Hidden-surface Algorithms," *Computing Surveys*, vol. 6, no. 1, March 1974.

17. J G Griffiths, "Tape-oriented hidden-line algorithm," *CAD*, vol. 13, no. 1, January 1981.

18. M Wittram, "Hidden-line algorithm for scenes of high complexity," *CAD*, vol. 13, no. 4, July 1981.

19. J R Woodwark and K M Quinlan, "Reducing the Effect of Complexity on Volume Model Evaluation," *CAD Journal*, vol. 4, no. 2, pp. 89-95, March 1982.

20. J E Warnock, "A Hidden Surface Algorithm for Computer-Generated Halftone Pictures," *Report TR 4-15, Computer Science Dept., University of Utah*, June 1969.

21. Atherton, "A scan-line hidden surface removal proceedure for constructive solid geometry," *Computer Graphics*, vol. 17, no. 3, July 1983.

22. W F Bronsvoort, "Techniques for reducing Booelan evaluation time in CSG scan-line algorithms," *CAD*, vol. 18, no. 10, December 1986.

23. Scott D Roth, "Ray Casting as a Method for Solid Modelling," *GM Research Publication*, vol. GMR-3466, General Motors Research Labs, Warren, Mitchigan 48090, October 1980.

24. W F Bronsvoort, J J van Wijk, and F W Jansen, "Two methods for improving the efficiency of ray casting in solid modelling," *CAD*, vol. 16, no. 1, January 1984.

25. W M Newman and R E Sproull, *Principles of Interactive Computer Graphics*, McGraw-Hill, 1979.

26. W A Hunt and H B Voelcker, "An Exploratory Study of Automatic Verification of Programs for Numnerically Controlled Machine Tools," *Production Automation Project Tech. Mem. 34*, The University of Rochester, Rochester, New York, January 1982.

27. J R Woodwark and K M Quinlan, "Derivation of Graphics from Volume Models by Recursive Division of the Object Space," *Proc. CG80, Brighton*, pp. 530-548, 1980.

28. K M Quinlan and J R Woodwark, "A Spatially-Segmented Solids Database - Justification and Design," *Proc. CAD82*, 1982.

29. K M Quinlan, "Utilising Spatial Locality in Solid Modelling," *PhD Thesis, Univerity of Bath*, 1985.

30. J R Woodwark and A Bowyer, "Better and Faster Pictures from Solid Models," *IEE Computer Aided Engineering Journal*, vol. 3, no. 2, 1986.

31. R B Tilove, A A Requicha, and M R Hopkins, "Efficient Editing of Solid Models by Exploiting Structural and Spatial Locality," *Production Automation Project Tech. Mem. 46*, The University of Rochester, Rochester, New York, May 1984.

32. N Okino, Y Kakazu, and H Kubo, "TIPS-1: Technical Information Processing System for Computer Aided Design, Drawing and Manufacturing," *Computer Languages for Numerical Control*, pp. North-Holland Publishing Company, 1973.

33. M Mañtyla and M Tamminen, "Localised set Operations for solid modelling," *Computer Graphics*, vol. 17, no. 3, July 1983.

34. M Tamminen, O Karonen, and M Mañtyla," "Ray-casting and block model conversion using spatial index," *CAD*, vol. 16, no. 4, July 1984.

35. G Markoswsky and M A Wesley, "Fleshing Out Wire Frames," *IBM Thomas J Watson Research Report*, no. RC 8124, 1980.

36. B Aldefeld, "On Automatic Recognition of 3D Structures from 2D Representations," *CAD Journal*, vol. 15, no. 2, March 1983.

37. K Preiss, "Constructing the 3-D Representation of a Plane-Faced Object from a Digitised Engineering Drawing," *Proc CAD80*, pp. 257-265, 1980.

38. I C Braid, "Designing with Volumes," *PhD Thesis, University of Cambridge*, 1973.

39. Shape Data Limited, Cambridge, England, *ROMULUS: Introduction*, Shape Data Limited, Cambridge, England, 1978.

40. A F Wallis and J R Woodwark, "Graphical Input to a Boolean Solid Modeller," *Proc CAD82*, March 1982.

41. D Peterson, "Halfspace Representation of Extrusions, Solids of Revolution, and Pyramids," *SANDIA Report*, no. SAND84-0572, Sandia National Labs, 1984.

42. S B Tor and A E Middleditch, "Convex Decomposition of Simple Polygons," *ACM Transactions on Graphics*, vol. 3, no. 4, pp. 244-265, October 1984.

43. D Peterson, "Boundary to Constructive Solid Geometry Mappings: a Focus on 2d Issues," *CAD*, vol. 18, no. 1, pp. 3-14, 1986.

44. Point Control, *SmartCAM Reference Manual*, Point Control Ltd, Eugene, Oregon, 1988.

45. B Chazelle and D Dobkin, "Decomposing a Polygon into its Convex Parts," *Proc. 11th Annual ACM Symp. on Theory of Computation, Atlanta*, pp. 38 - 48, 1979.

46. P J Green and B W Silverman, "Constructing the Convex Hull of a Set of Points in the Plane," *Computer Journal*, vol. 22, no. 3, pp. 262 - 266, 1980.

47. R A Jarvis, "On the Identification of the Convex Hull of a Finite Planar Set," *Information Processing Letters*, vol. 2, pp. 18 - 21, 1973.

48. R G Francis, *Graphical Numerical Control*, p. National Engineering Laboratory, Glasgow, CAD Centre, 1978.

49. *Computer Assistance in Tape Proving*, NEL, 1975.

50. P McGoldrick and R Gibson, "NC plotting made simple," *Proc. CAD80*, 1980.

51. R O Anderson, "Detecting and Eliminations Collisions in NC Machining," *CAD*, vol. 12, no. 4, pp. 231-237, July 1978.

52. I T Chappel, "The use of vectors to simulate material removed by numerically controlled milling," *CAD*, vol. 15, no. 3, May 1983.

53. T van Hook, "Realtime shaded nc milling display," *Computer Graphics*, vol. 20, no. 4, pp. 15-20, 1986.

54. R Fridshal, K P Cheng, D Duncan, and W Zucker, "Numerical Control PArt Program Verification System," *Proc. Conference on CAD/CAM Technology in Mechanical Engineering, MIT*, 1982.

55. R R Martin and P C Stephenson, "Arbitrary Sweeping of Three-Dimensional Objects," *Submitted to CADJ, August 1989.* (in press)

56. P Simkins, *Description of CLdata records produced by GNC*, CAD Centre, 1979.

57. IIT Research Institute, in *APT Part Programming*, McGraw Hill, 1967.

58. A F Wallis and J R Woodwark, "Creating Large Solid Models for N.C. Toolpath Verification," *Proc. CAD84*, April 1984.

59. R E Moore, "Methods and Applications of Interval Analysis," *Proc. SIAM Conf 1979*, 1979.

60. A F Wallis and J R Woodwark, "Interrogating Solid Models," *Proc. CAD84*, April 1984.